CHAPTER 5

CRITICAL CHOICES: what when and how to test

- RISK AND RISK MANAGEMENT
- START TESTING EARLY
- BASIC FORMS OF TESTING PROCESS
- TESTING,THE DEVELOPMENT CYCLE,REAL WORLD OF CONTACTS
- EFFECTIVE TESTING AND COST EFFECTIVE TESTING

Introduction

- **Exhaustive testing** mean that product never reach the market because testing would never be completed.
- **Fifty percent** or more time of developmental organization is spent on error detection and removal.
- One choice is **test the right things-** ensure more critical items are tested don’t waste resource on less important items.
- **Test early**: Prevent costly migration
- Testing life cycle include verification testing and validation testing

- Critical items are tested not waste resource on limited resources , error are closure to the phase where they are introduced
- Costly migration of error downstream.

Risk and risk management

Risk is the probability that undesirable things will happen, such as loss of human life, or large financial losses. The systems we develop, when they don't work properly, have consequences that can vary from the mildly irritating to the catastrophic. Testing these systems should involve informed, conscious risk management.

We can't do everything. We have to make compromises, but we don't want to take risks that are unacceptably high. Key questions must be asked:

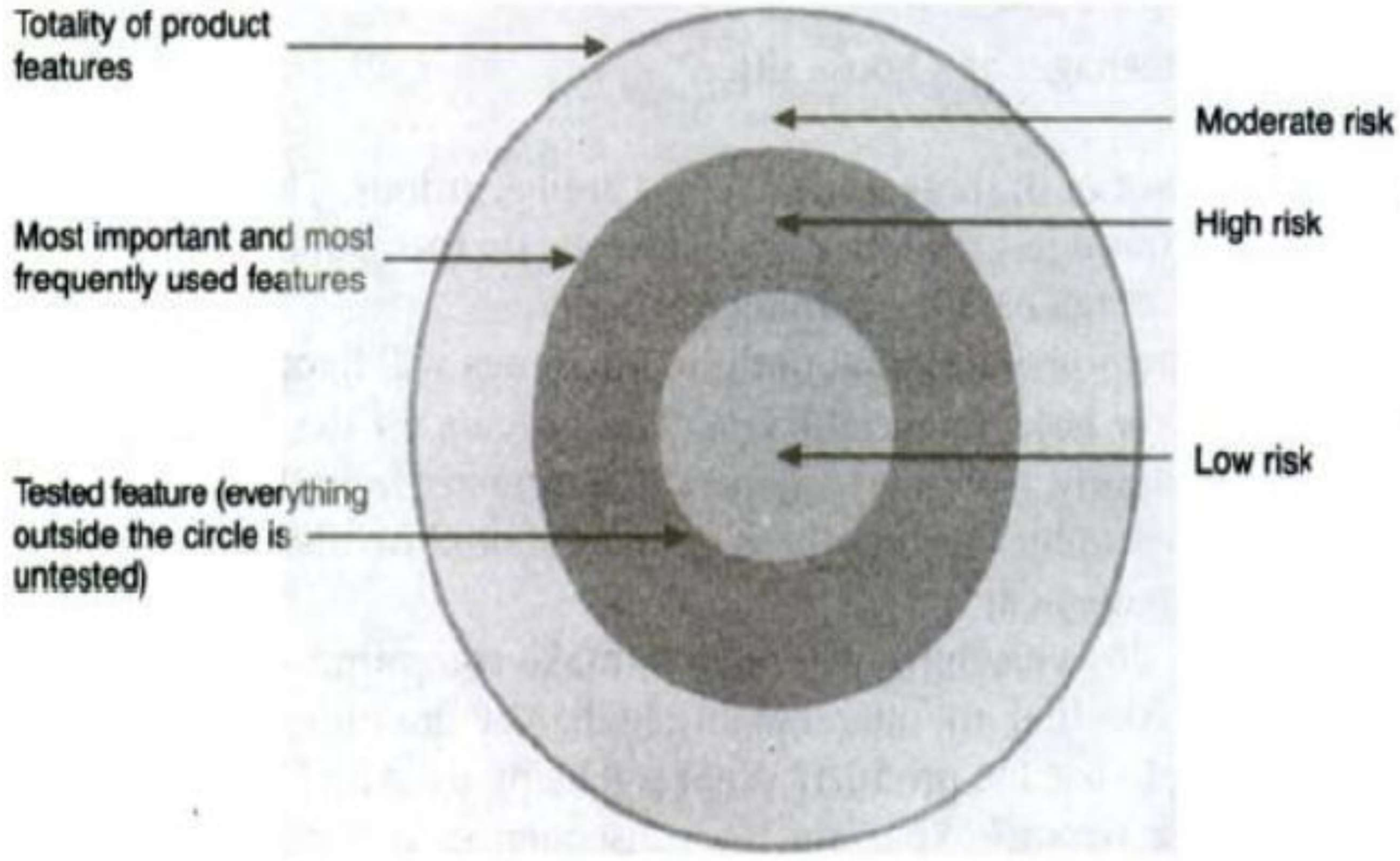
"Who is going to use the product? What is it being used for? What is the danger of it going wrong? What are the consequences if it does? Is it loss of money? Is it a loss of customer satisfaction? Is it loss of life?"

For each product, we must implement the most cost-effective testing that will ensure that it's reliable enough, safe enough and meets the user/customer requirement.

Another basis for choice of testing focus is frequency of use. If a part of the system is used often, and it has an error in it, its frequent use alone makes the chances of a failure emerging much higher.

For each testing activity to be performed, testing objectives are prioritized by making judgments on the potential for all kinds of failure situations.

Risk is not just the basis for making management decisions on testing. It is the basis for decisions that test practitioners make every day.


Totality of product features
Moderate risk
High risk
Most important and most frequently used features
Low risk
Tested feature (everything outside the circle is untested)

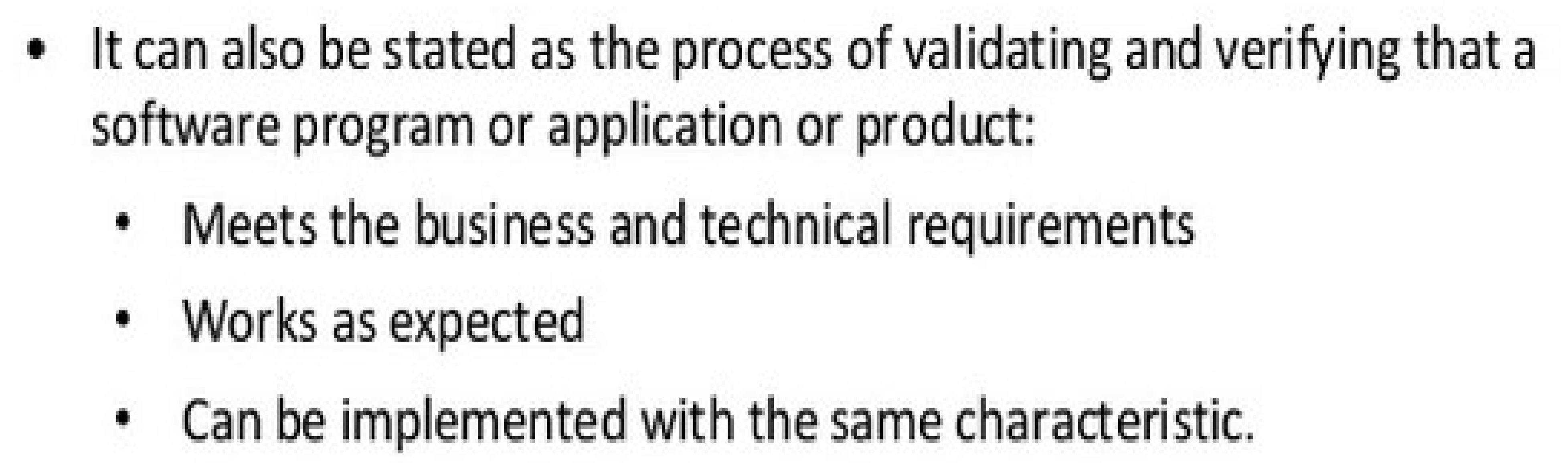
Risk in system as a basis for testing choices

Start testing early

- Test early and prevent defect migration.
- Find a defect. If we don’t find it, it’s allowed to migrate.
- In the real world of projects under pressure & in a less than ideal organization, What can we do about poor reqmts? They can be verified using the review methods

BASIC FORMS OF TESTING:

- Verification
- Validation



- It can also be stated as the process of validating and verifying that a software program or application or product:
  - Meets the business and technical requirements
  - Works as expected
  - Can be implemented with the same characteristic.

1.1 Verification and Validation Basic Concepts

- Verification

  "**Are we building the product right**"

- **The software should conform to its <u>specification</u>**

- Validation

  "**Are we building the right product**"

- **The software should do what the user really requires.**
- **i.e. is the customer need <u>(requirements)</u> met?**

Verification, as defined by IEEE/ANSI, is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

Some people call verification "human" testing, because it usually involves looking at documents on paper. By contrast, validation usually involves the execution of software on a computer.

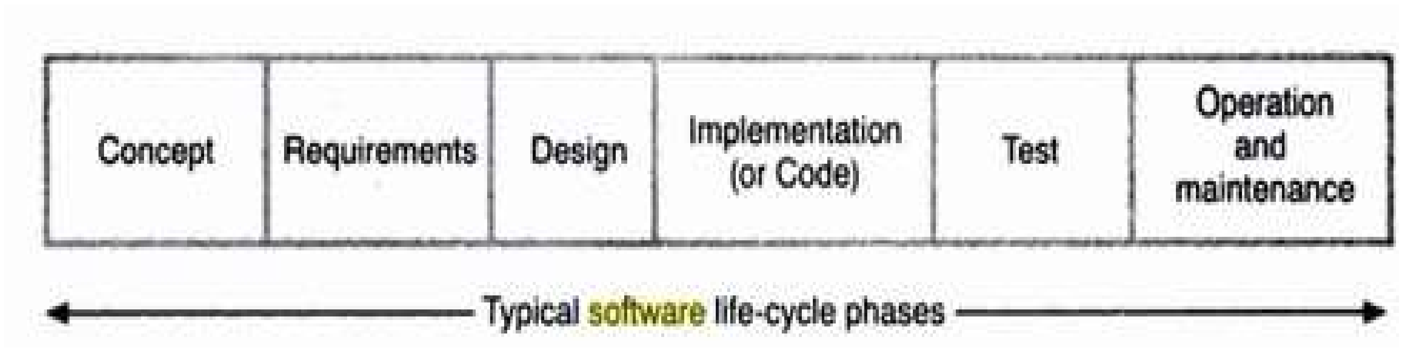
Validation, as defined by IEEE/ANSI, is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

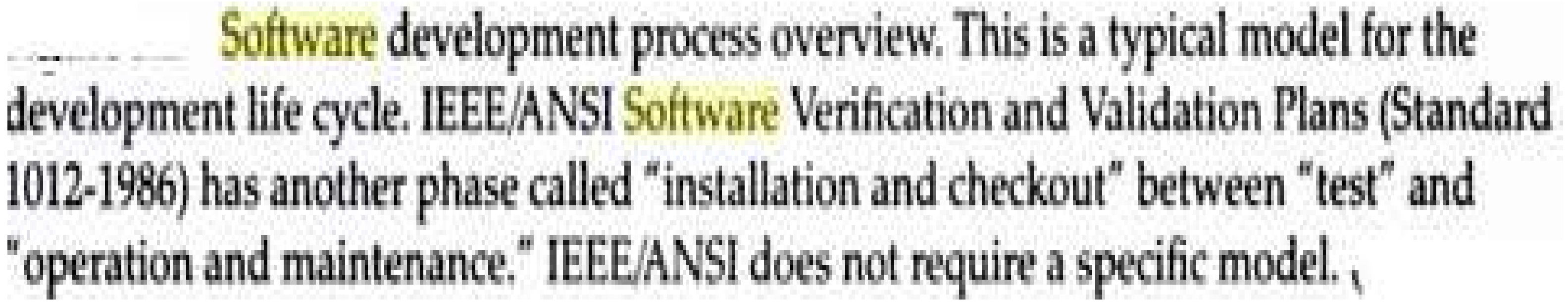
Validation normally involves executing the actual software or a simulated mock-up. Validation is a "computer-based testing" process. It usually exposes symptoms of errors.

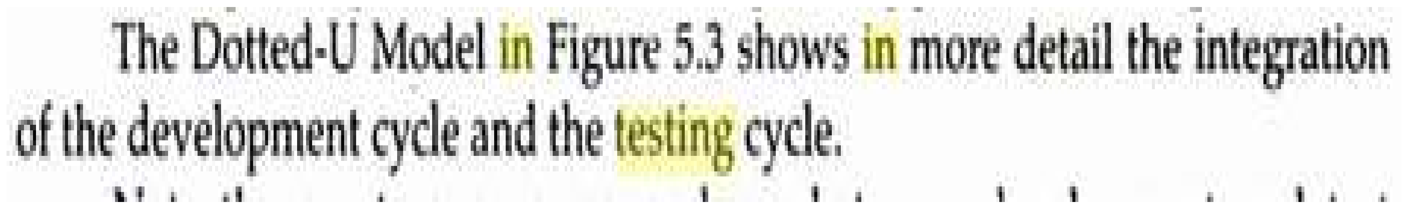
It is a central thesis of this book that testing includes both verification and validation.

Definition: Testing = verification plus validation

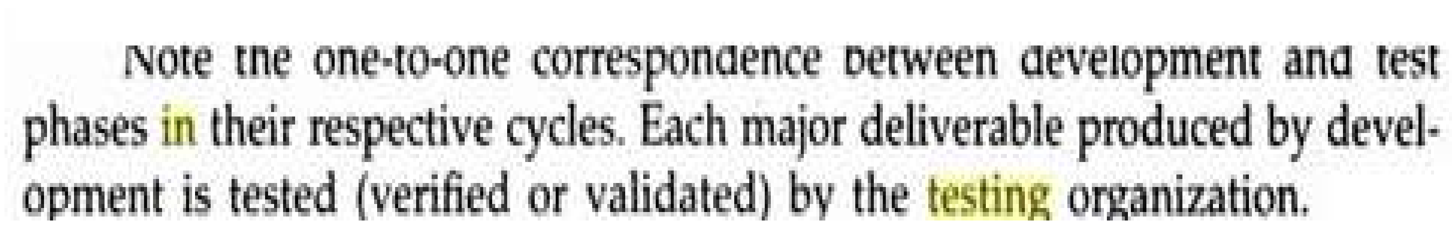
TESTING,THE DEVELOPMENT CYCLE,REAL WORLD OF CONTACTS



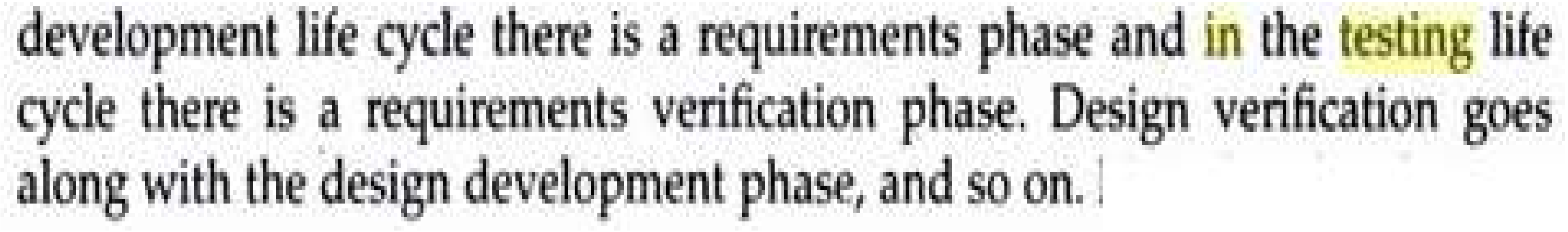

Software development process overview. This is a typical model for the development life cycle. IEEE/ANSI Software Verification and Validation Plans (Standard 1012-1986) has another phase called "installation and checkout" between "test" and "operation and maintenance." IEEE/ANSI does not require a specific model.



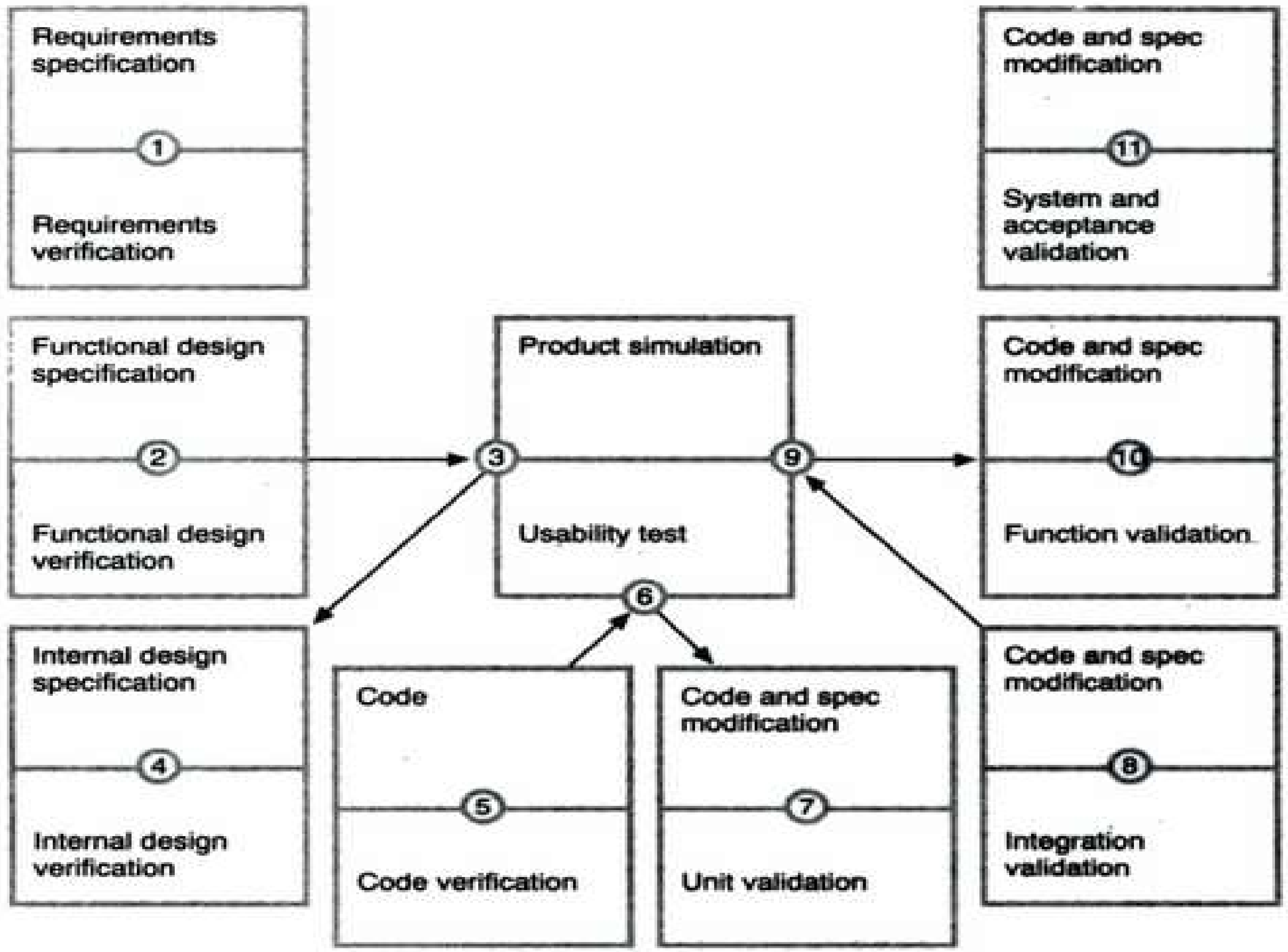
The Dotted-U Model in Figure 5.3 shows in more detail the integration of the development cycle and the testing cycle.

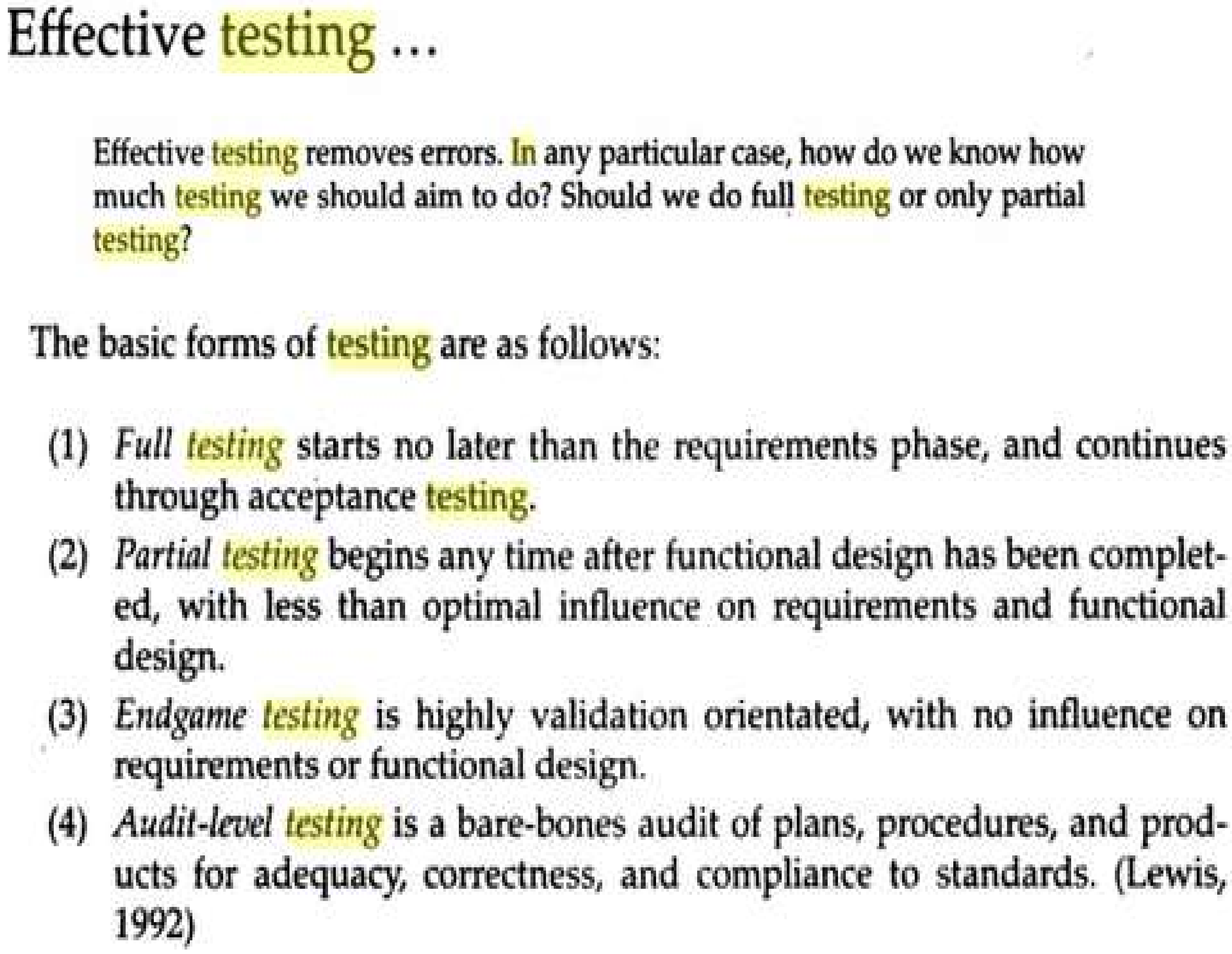


Note the one-to-one correspondence between development and test phases in their respective cycles. Each major deliverable produced by development is tested (verified or validated) by the testing organization.



development life cycle there is a requirements phase and in the testing life cycle there is a requirements verification phase. Design verification goes along with the design development phase, and so on.


Requirements specification
1
Requirements verification
Functional design specification
2
Functional design verification
Internal design specification
4
Internal design verification
Product simulation
3
9
Usability test
6
Code
5
Code verification
Code and spec modification
7
Unit validation
Code and spec modification
8
Integration validation
Code and spec modification
10
Function validation
Code and spec modification
11
System and acceptance validation



Effective testing …

Effective testing removes errors. In any particular case, how do we know how much testing we should aim to do? Should we do full testing or only partial testing?

The basic forms of testing are as follows:

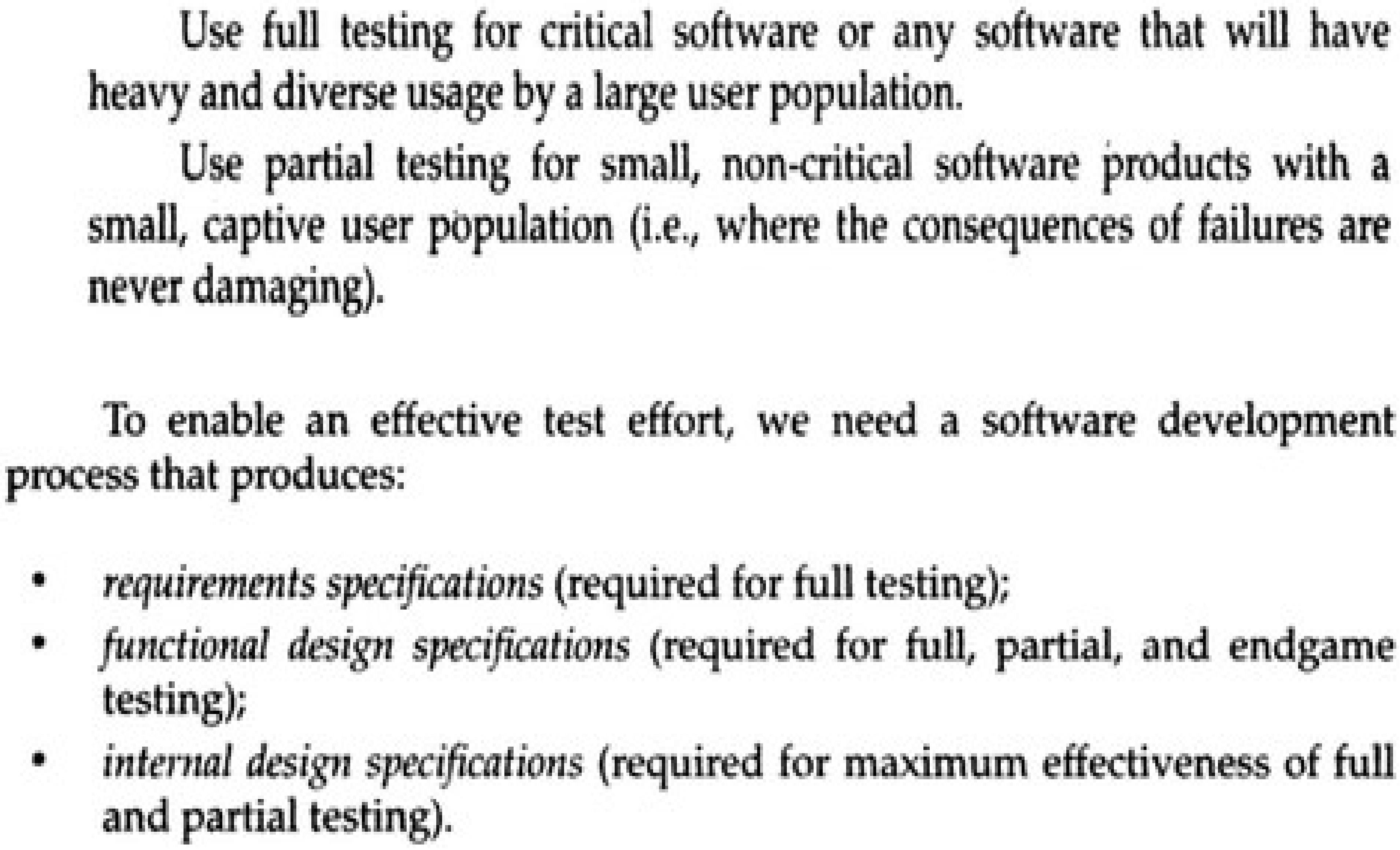
(1) *Full testing* starts no later than the requirements phase, and continues through acceptance testing.

(2) *Partial testing* begins any time after functional design has been completed, with less than optimal influence on requirements and functional design.

(3) *Endgame testing* is highly validation orientated, with no influence on requirements or functional design.

(4) *Audit-level testing* is a bare-bones audit of plans, procedures, and products for adequacy, correctness, and compliance to standards. (Lewis, 1992)

- Critical software is software whose failure cold have **impact on safety**, or could cause large **financial or social loss**.

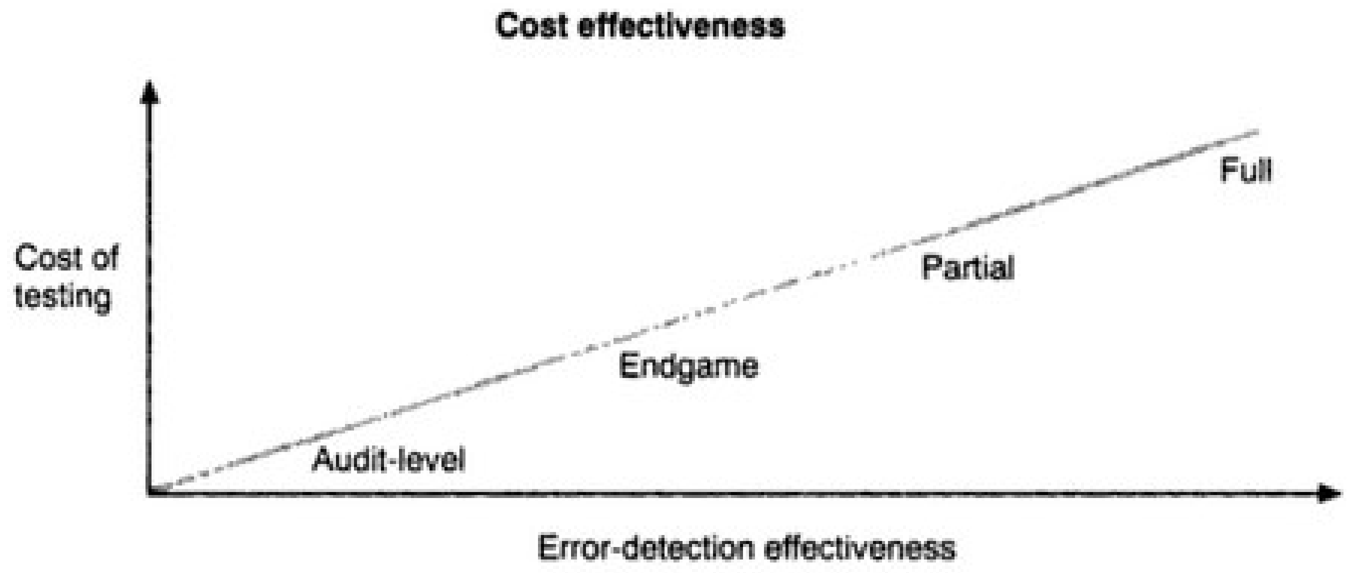


Use full testing for critical software or any software that will have heavy and diverse usage by a large user population.

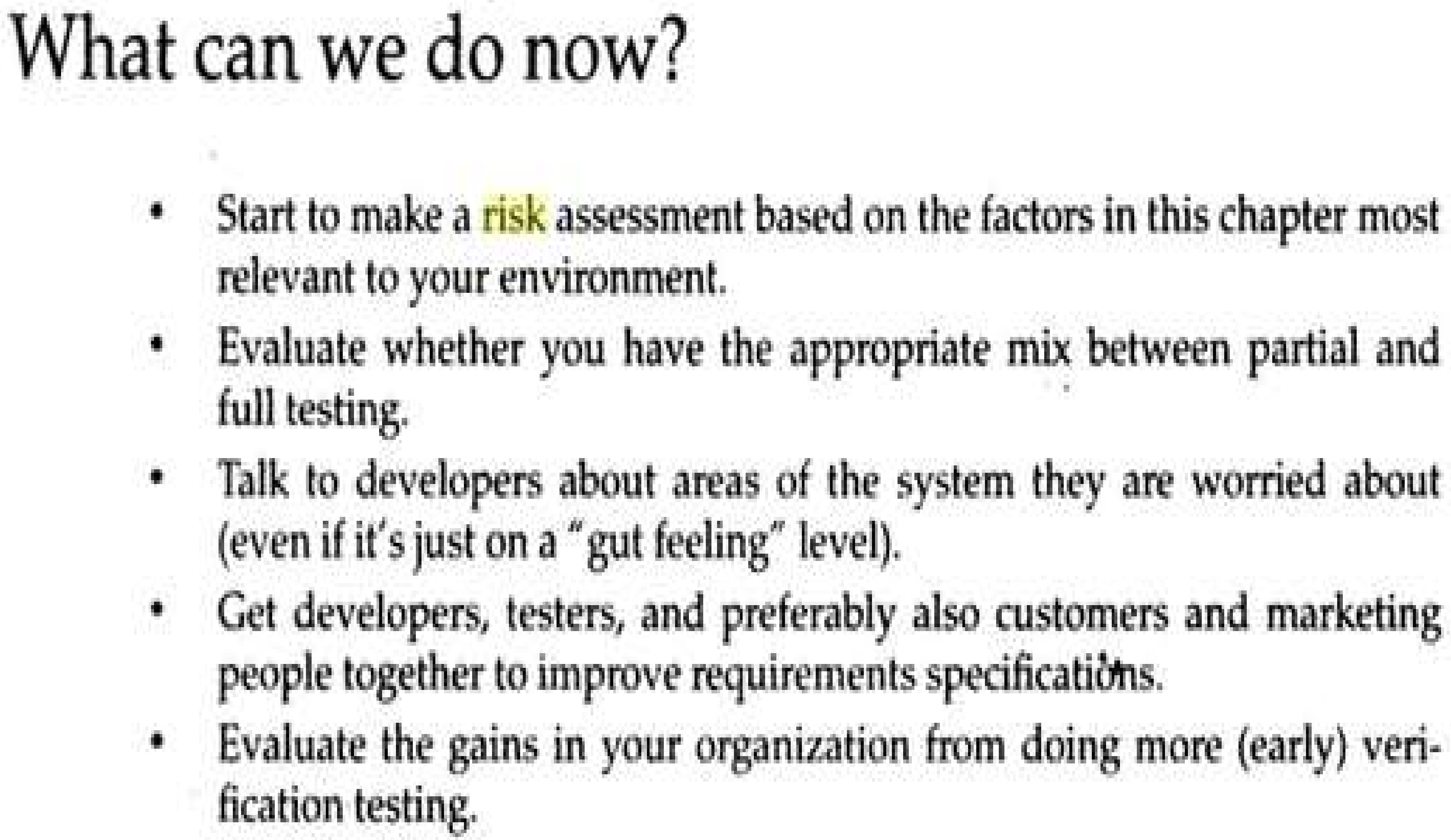
Use partial testing for small, non-critical software products with a small, captive user population (i.e., where the consequences of failures are never damaging).

To enable an effective test effort, we need a software development process that produces:

- *requirements specifications* (required for full testing);
- *functional design specifications* (required for full, partial, and endgame testing);
- *internal design specifications* (required for maximum effectiveness of full and partial testing).



The more effective the error detection, the greater the savings in development and maintenance costs over the life of the product. (© 1993, 1994 Software Development Technologies)



What can we do now?

- Start to make a risk assessment based on the factors in this chapter most relevant to your environment.
- Evaluate whether you have the appropriate mix between partial and full testing.
- Talk to developers about areas of the system they are worried about (even if it's just on a "gut feeling" level).
- Get developers, testers, and preferably also customers and marketing people together to improve requirements specifications.
- Evaluate the gains in your organization from doing more (early) verification testing.

Chapter 6 Critical disciplines: Framework for testing

Planning

- Considerations in verification planning
- Considerations in verification planning

Software Engineering Maturity and SEI Process Maturity levels:

- How process maturity level affects testing

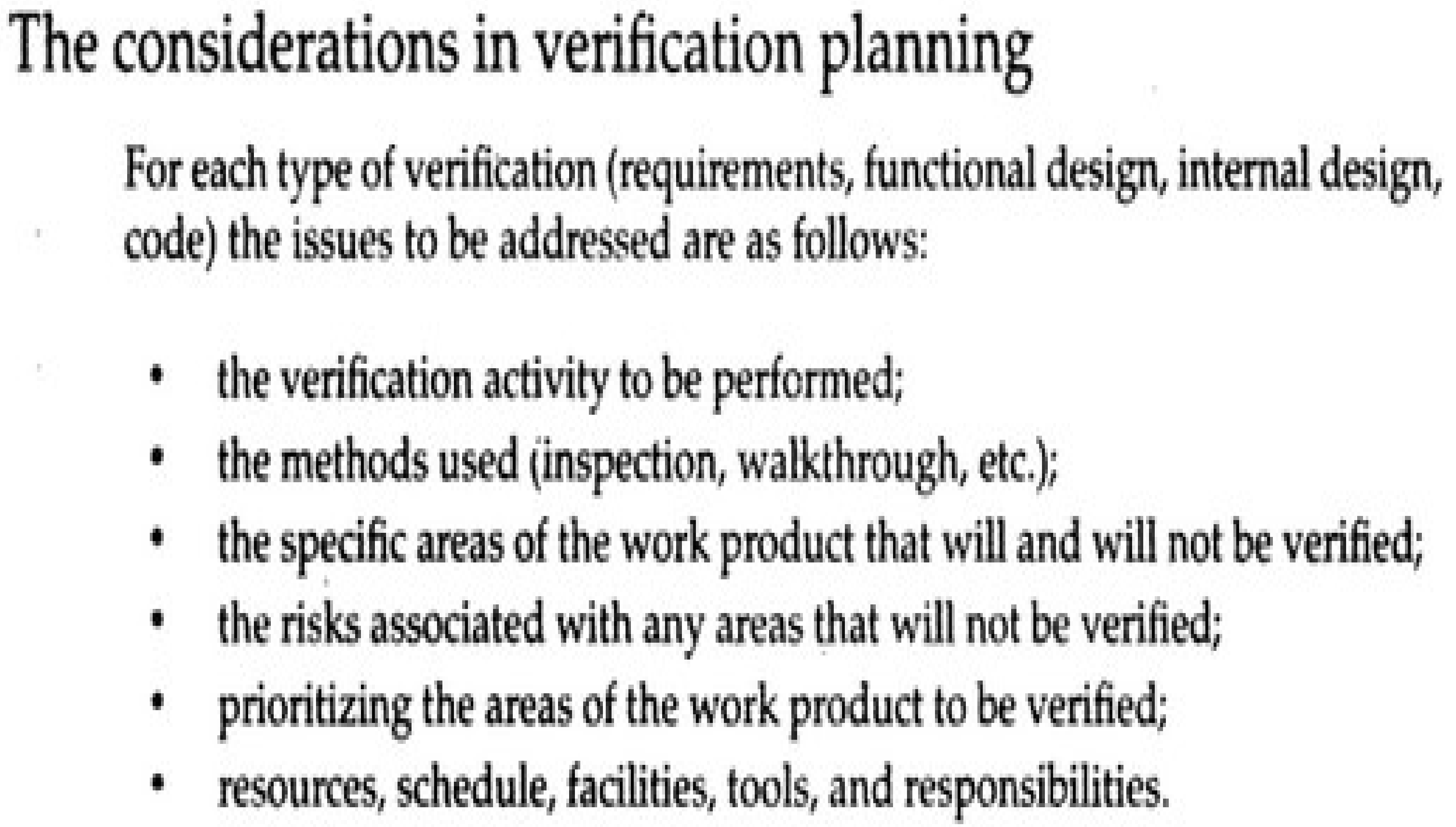
Configuration Management

- Testing Interest in configuration management

- Standards
  - IEEE/ANSI Standards
- Formal documents
- Measurement
- Tools

Planning

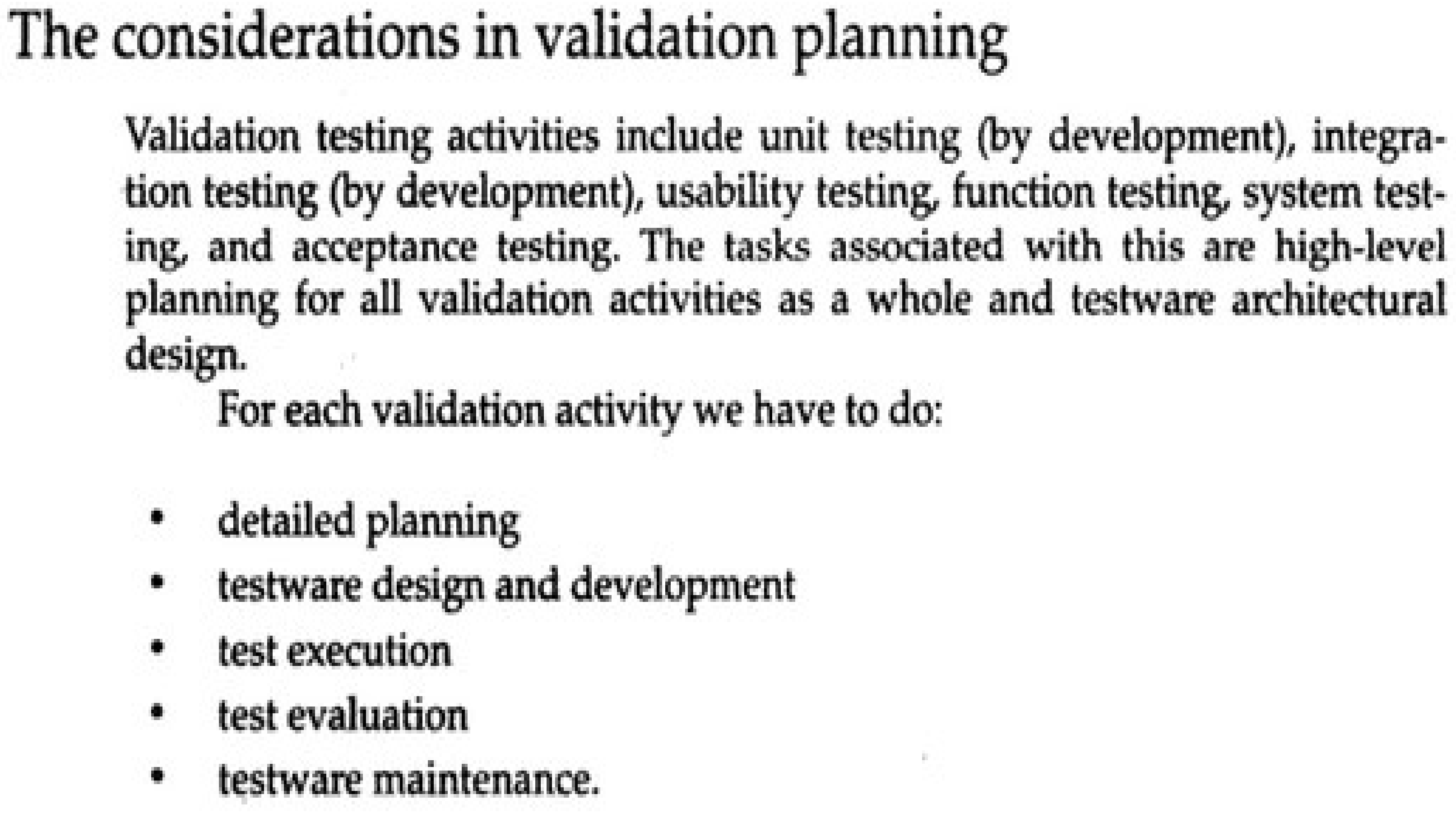
- First step of activity
- Review requirements before testing
- Improve specification by good testing.
- What kind of standards are going to useful?
- Inspections and reviews?



The considerations in verification planning

For each type of verification (requirements, functional design, internal design, code) the issues to be addressed are as follows:

- the verification activity to be performed;
- the methods used (inspection, walkthrough, etc.);
- the specific areas of the work product that will and will not be verified;
- the risks associated with any areas that will not be verified;
- prioritizing the areas of the work product to be verified;
- resources, schedule, facilities, tools, and responsibilities.



The considerations in validation planning

Validation testing activities include unit testing (by development), integration testing (by development), usability testing, function testing, system testing, and acceptance testing. The tasks associated with this are high-level planning for all validation activities as a whole and testware architectural design.

For each validation activity we have to do:

- detailed planning
- testware design and development
- test execution
- test evaluation
- testware maintenance.

For validation planning the issues to be addressed are as follows:

- test methods
- facilities (for testware development vs. test execution)
- test automation
- testing tools
- support software (shared by development and test)
- configuration management
- risks (budget, resources, schedule, training).

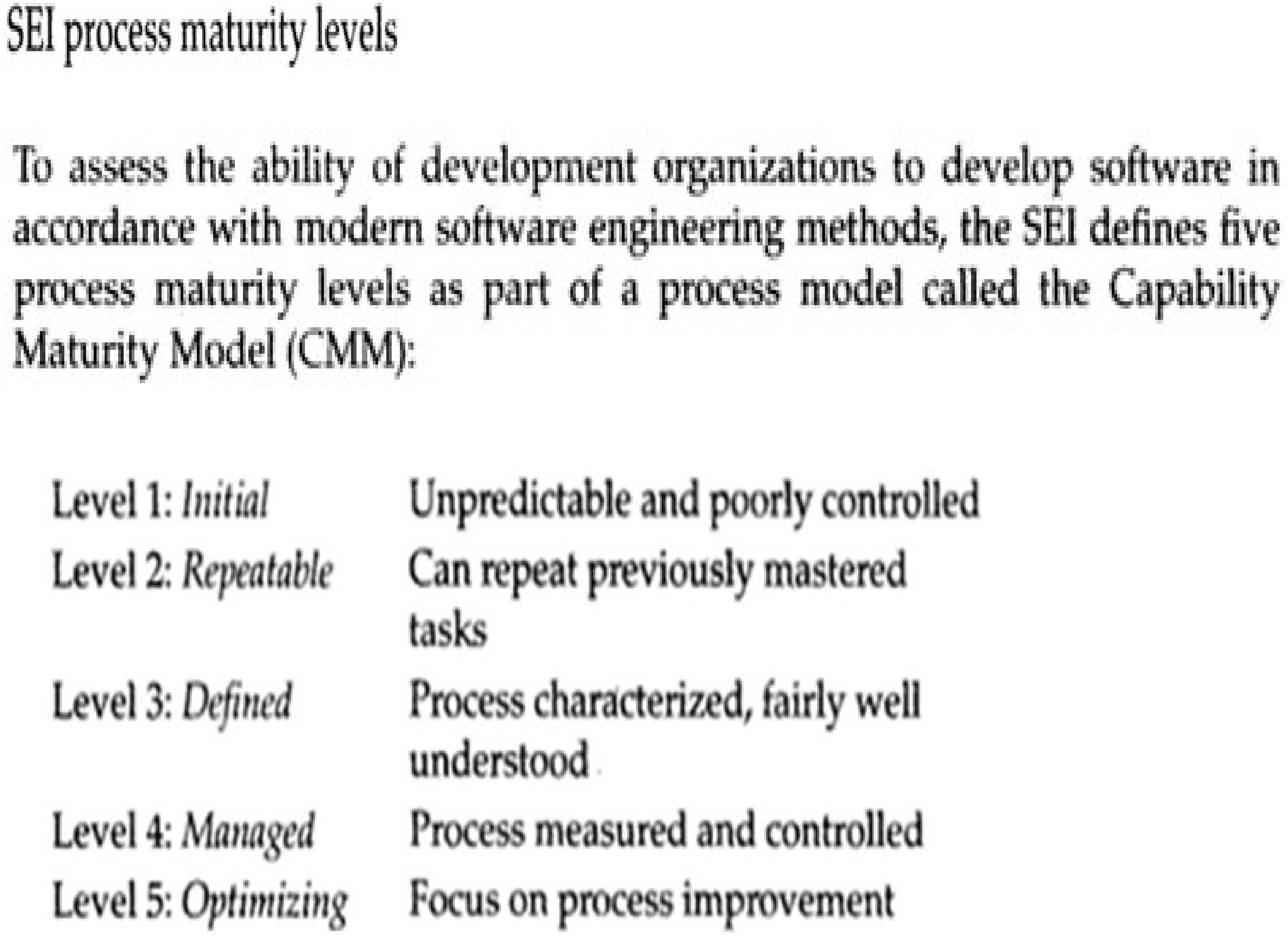
Software engineering maturity and SEI

The Software Engineering Institute (SEI) is a federally-funded research and development center, sponsored by the Department of Defense, and affiliated with Carnegie Mellon University.

address a twofold shortage of trained software professionals and quality software, produced on schedule and within budget.

SEI focuses on software process,

The mission of the SEI is to provide leadership to advance the state-of-the-practice of software engineering to improve the quality of systems that depend on software.



SEI process maturity levels

To assess the ability of development organizations to develop software in accordance with modern software engineering methods, the SEI defines five process maturity levels as part of a process model called the Capability Maturity Model (CMM):

Level 1: *Initial*	Unpredictable and poorly controlled
Level 2: *Repeatable*	Can repeat previously mastered tasks
Level 3: *Defined*	Process characterized, fairly well understood
Level 4: *Managed*	Process measured and controlled
Level 5: *Optimizing*	Focus on process improvement

Level 3, Activity 5: Software testing is performed according to the project's defined software process

Level 3, Activity 6: Integration testing of the software is planned and performed according to the project's defined software process

Level 3, Activity 7: System and acceptance testing of the software are planned and performed to demonstrate that the software satisfies its requirements

Level 3, Activity 5: Software testing is performed according to the project's defined software process

(1) Testing criteria are developed and reviewed with the customer and the end user, as appropriate.

(2) Effective methods are used to test the software.

(3) The adequacy of testing is determined by:

  (i) the level of testing performed (e.g., unit testing, integration testing, system testing),

  (ii) the test strategy selected (e.g., black-box, white-box), and

  (iii) the test coverage to be achieved (e.g., statement coverage, branch coverage).

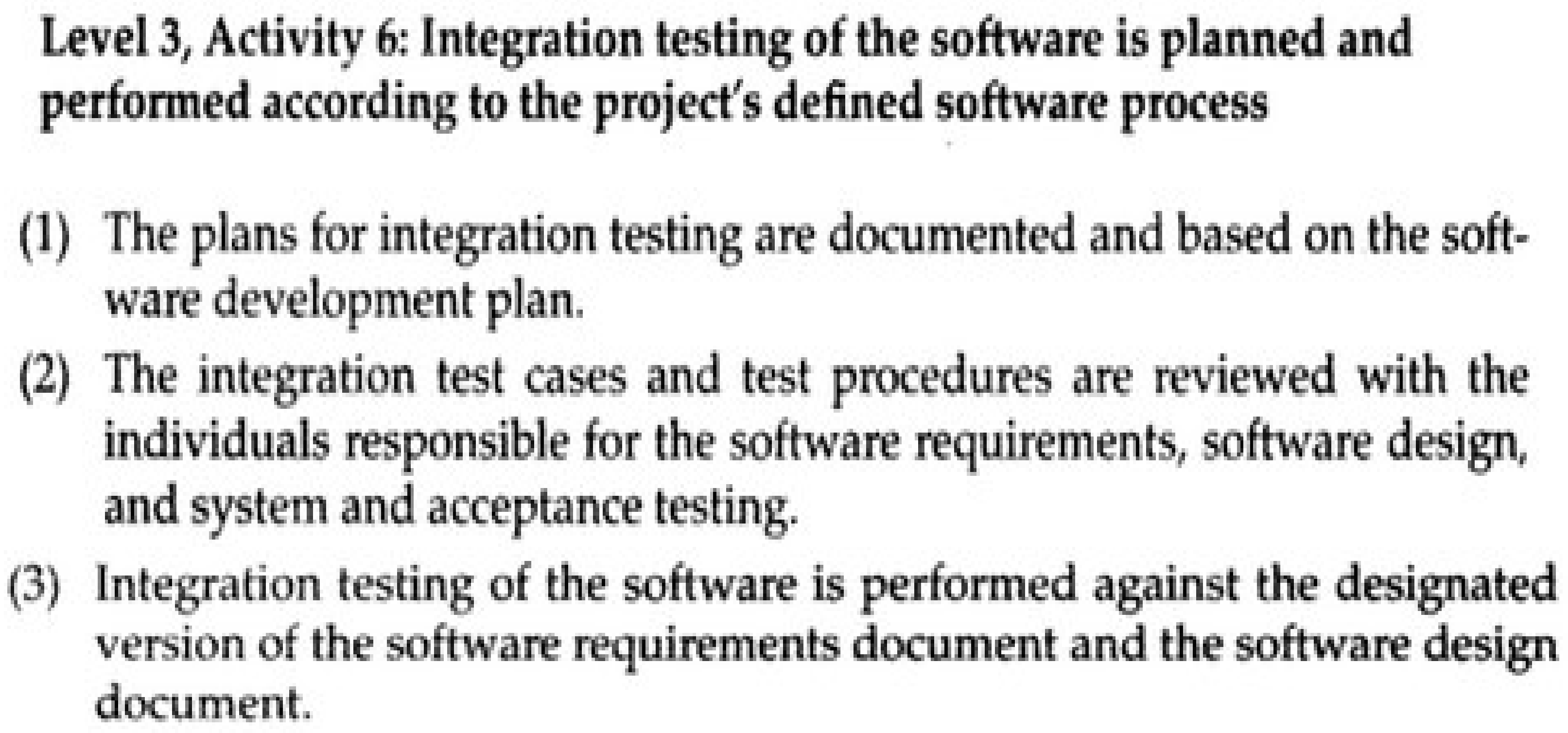
(4) For each level of software testing, test readiness criteria are established and used.

(5) Regression testing is performed, as appropriate, at each test level whenever the software being tested or its environment changes.

(6) The test plans, test procedures, and test cases undergo peer review before they are considered ready for use.

(7) The test plans, test procedures, and test cases are managed and controlled (e.g., placed under configuration management).

(8) Test plans, test procedures, and test cases are appropriately changed whenever the allocated requirements, software requirements, software design, or code being tested changes.

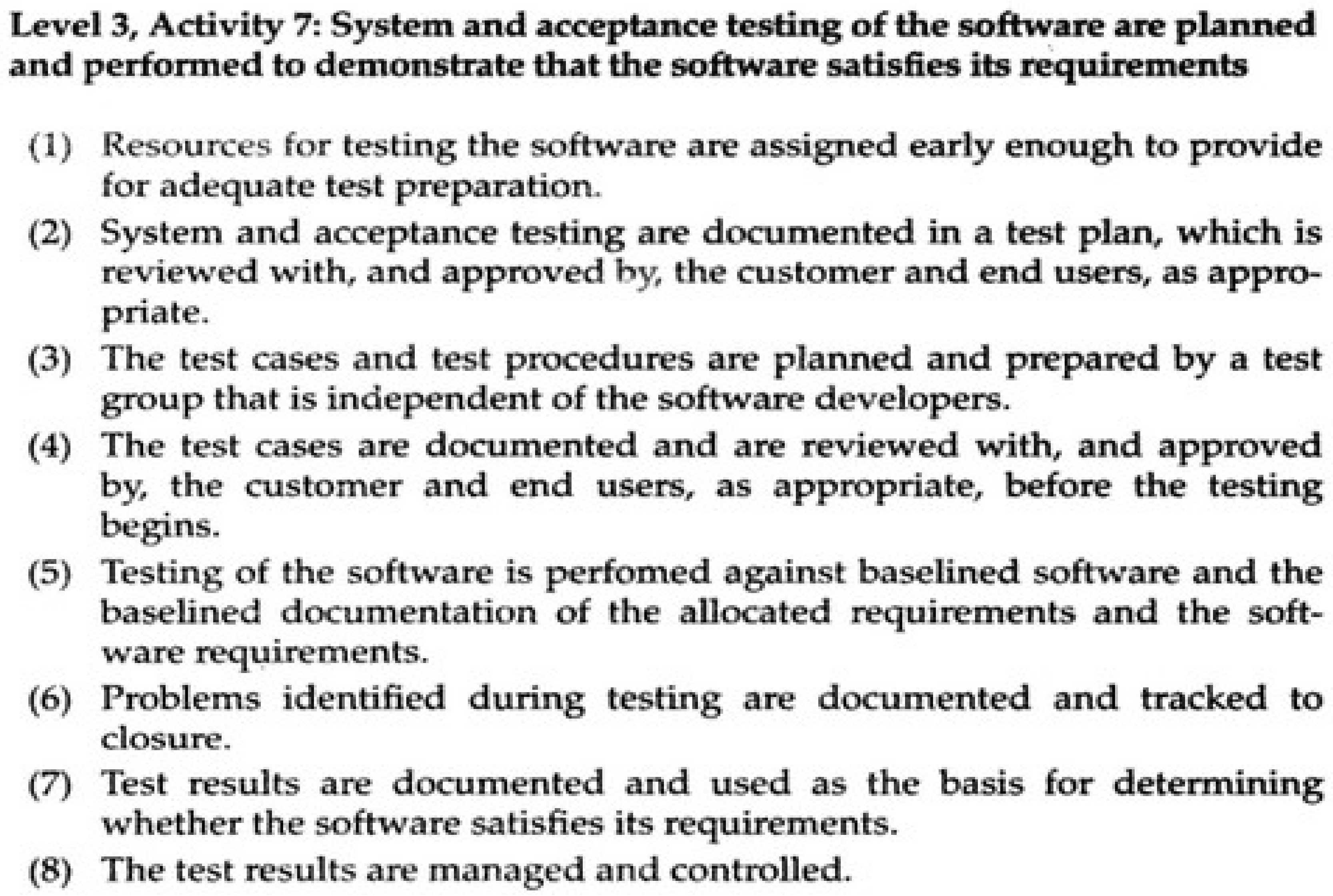


Level 3, Activity 6: Integration testing of the software is planned and performed according to the project's defined software process

(1) The plans for integration testing are documented and based on the software development plan.

(2) The integration test cases and test procedures are reviewed with the individuals responsible for the software requirements, software design, and system and acceptance testing.

(3) Integration testing of the software is performed against the designated version of the software requirements document and the software design document.



Level 3, Activity 7: System and acceptance testing of the software are planned and performed to demonstrate that the software satisfies its requirements

(1) Resources for testing the software are assigned early enough to provide for adequate test preparation.

(2) System and acceptance testing are documented in a test plan, which is reviewed with, and approved by, the customer and end users, as appropriate.

(3) The test cases and test procedures are planned and prepared by a test group that is independent of the software developers.

(4) The test cases are documented and are reviewed with, and approved by, the customer and end users, as appropriate, before the testing begins.

(5) Testing of the software is perfomed against baselined software and the baselined documentation of the allocated requirements and the software requirements.

(6) Problems identified during testing are documented and tracked to closure.

(7) Test results are documented and used as the basis for determining whether the software satisfies its requirements.

(8) The test results are managed and controlled.

How process maturity affects testing

- amount of testing budget spent on non technical issues is inverse proportional to maturity level within environment as a whole.
- Level 1: uncertainty outcome is unpredictable
- SPA :software process assessment-special consultancy service.
- Evaluating s/w process maturity

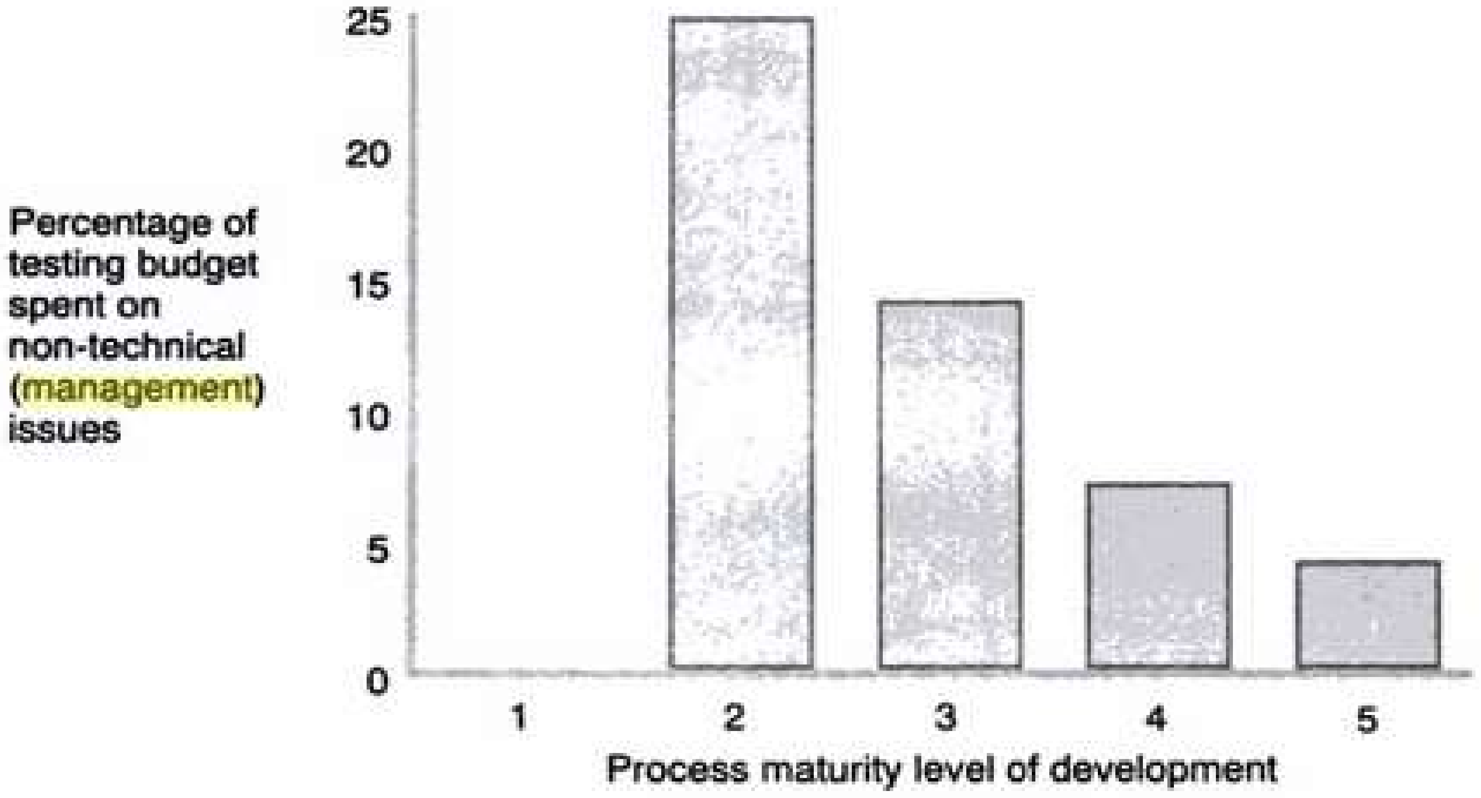
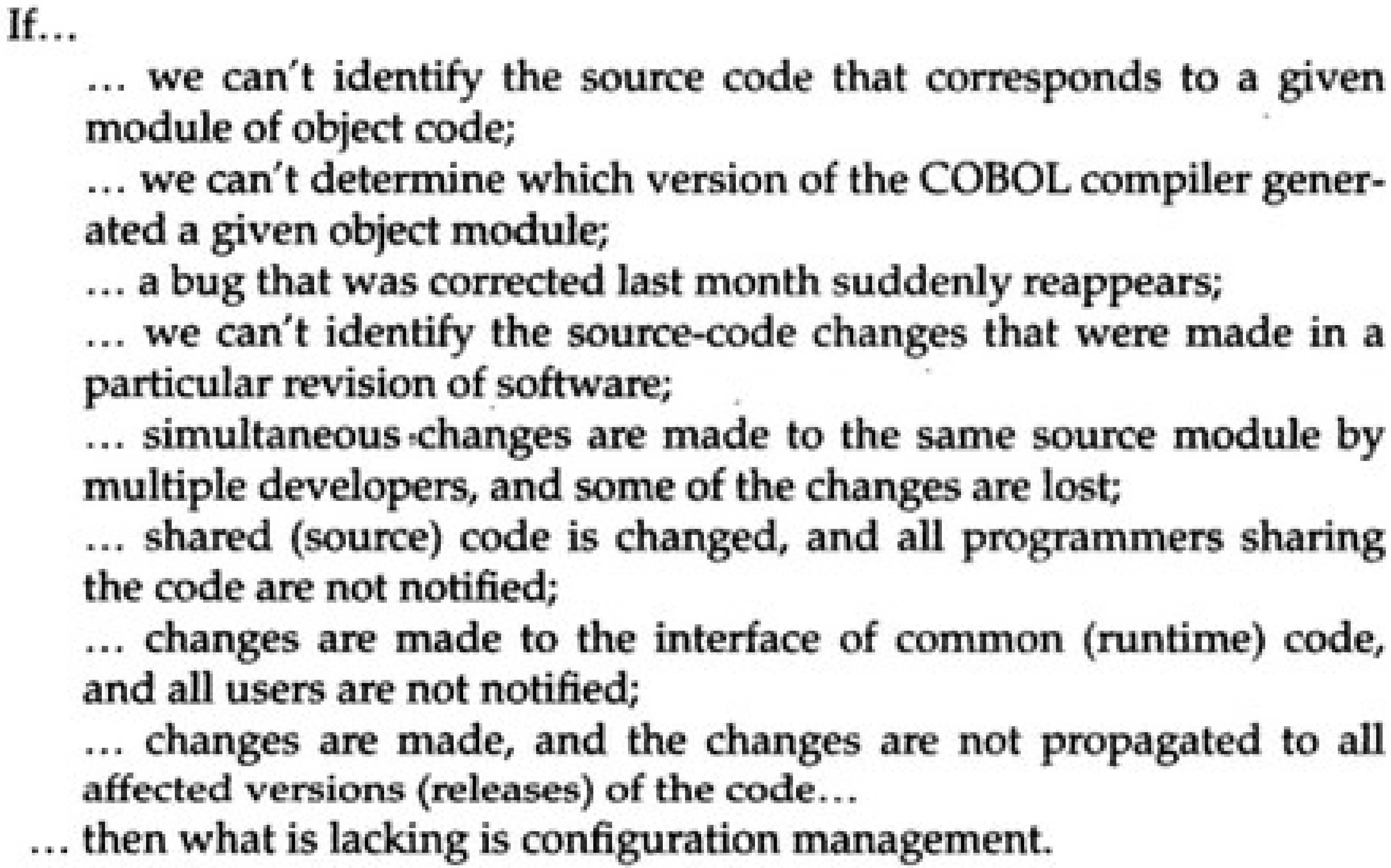


Figure 6.1 How the process maturity level affects testing (Lewis, 1992). To the extent that development doesn't know where it is going, testing cannot plan its own activities and will spend more time on management issues rather than technical issues. (© 1993, 1994 Software Development Technologies)

- **Configuration management** (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life.

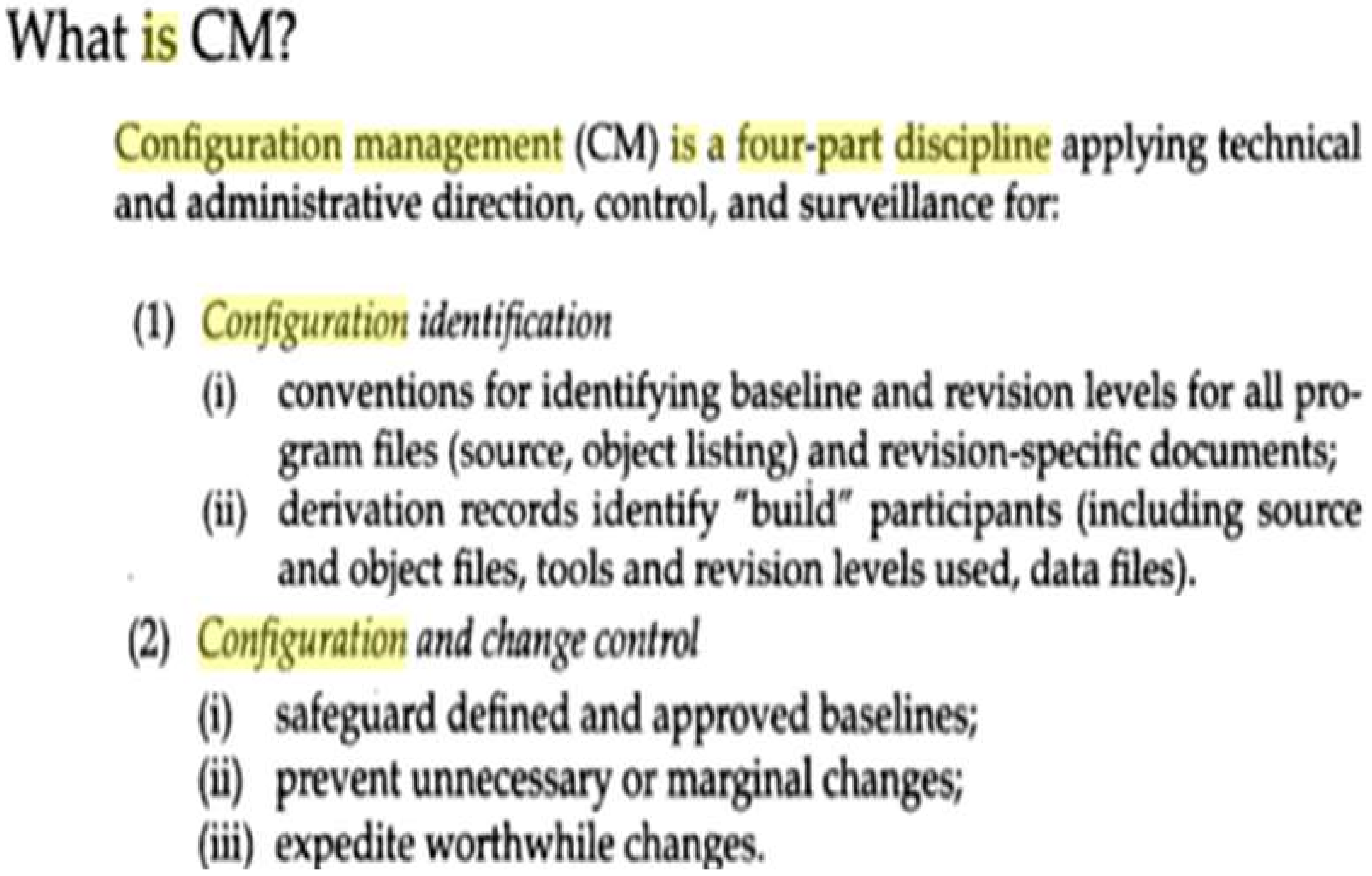


If…

… we can't identify the source code that corresponds to a given module of object code;
… we can't determine which version of the COBOL compiler generated a given object module;
… a bug that was corrected last month suddenly reappears;
… we can't identify the source-code changes that were made in a particular revision of software;
… simultaneous changes are made to the same source module by multiple developers, and some of the changes are lost;
… shared (source) code is changed, and all programmers sharing the code are not notified;
… changes are made to the interface of common (runtime) code, and all users are not notified;
… changes are made, and the changes are not propagated to all affected versions (releases) of the code…

… then what is lacking is configuration management.

- **Configuration management** (CM) is a systems engineering process for establishing and maintaining consistency of a product's performance, functional, and physical attributes with its requirements, design, and operational information throughout its life.



What is CM?

Configuration management (CM) is a four-part discipline applying technical and administrative direction, control, and surveillance for:

(1) *Configuration identification*

(i) conventions for identifying baseline and revision levels for all program files (source, object listing) and revision-specific documents;

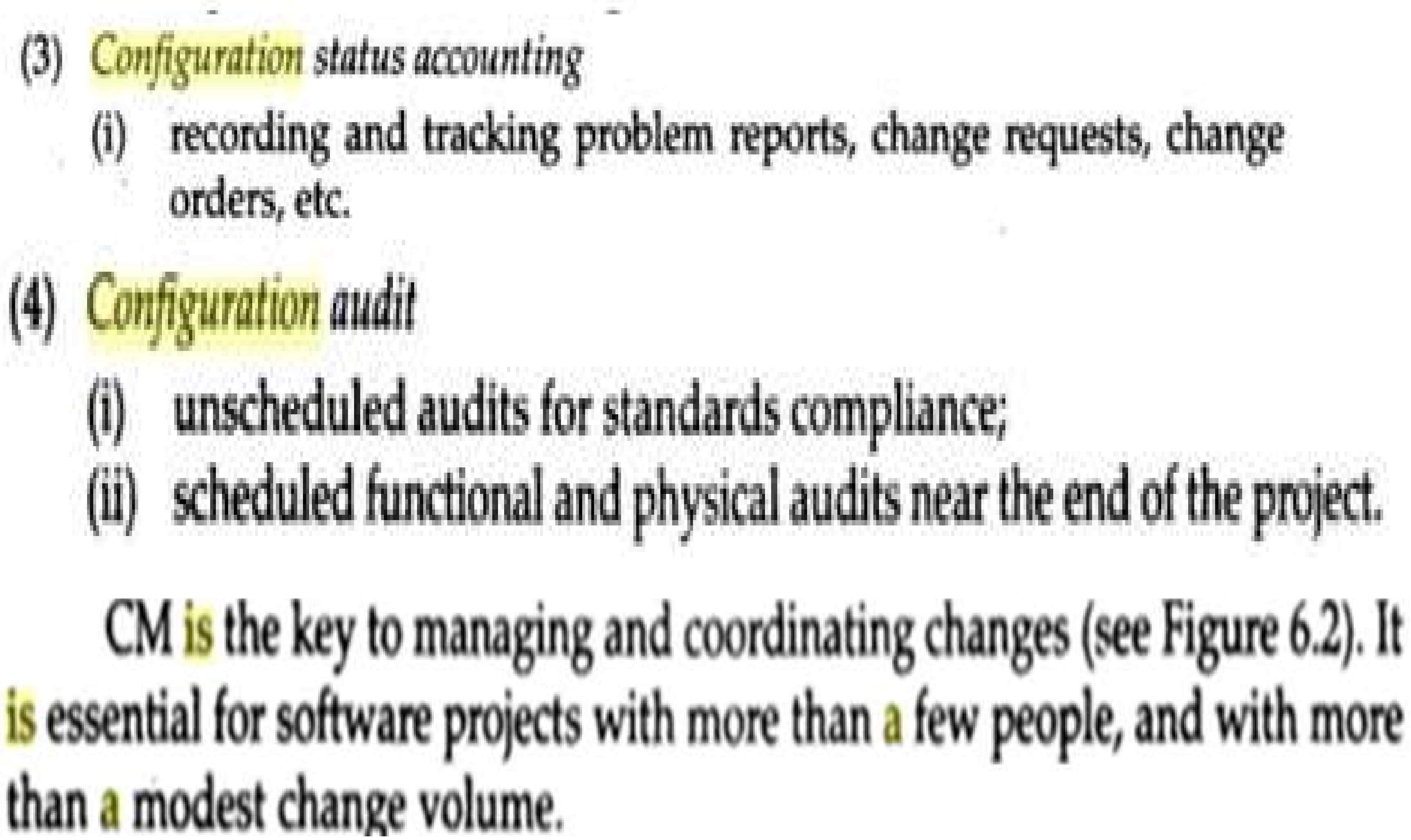
(ii) derivation records identify "build" participants (including source and object files, tools and revision levels used, data files).

(2) *Configuration and change control*

(i) safeguard defined and approved baselines;

(ii) prevent unnecessary or marginal changes;

(iii) expedite worthwhile changes.



(3) *Configuration status accounting*

(i) recording and tracking problem reports, change requests, change orders, etc.

(4) *Configuration audit*

(i) unscheduled audits for standards compliance;

(ii) scheduled functional and physical audits near the end of the project.

CM is the key to managing and coordinating changes (see Figure 6.2). It is essential for software projects with more than a few people, and with more than a modest change volume.

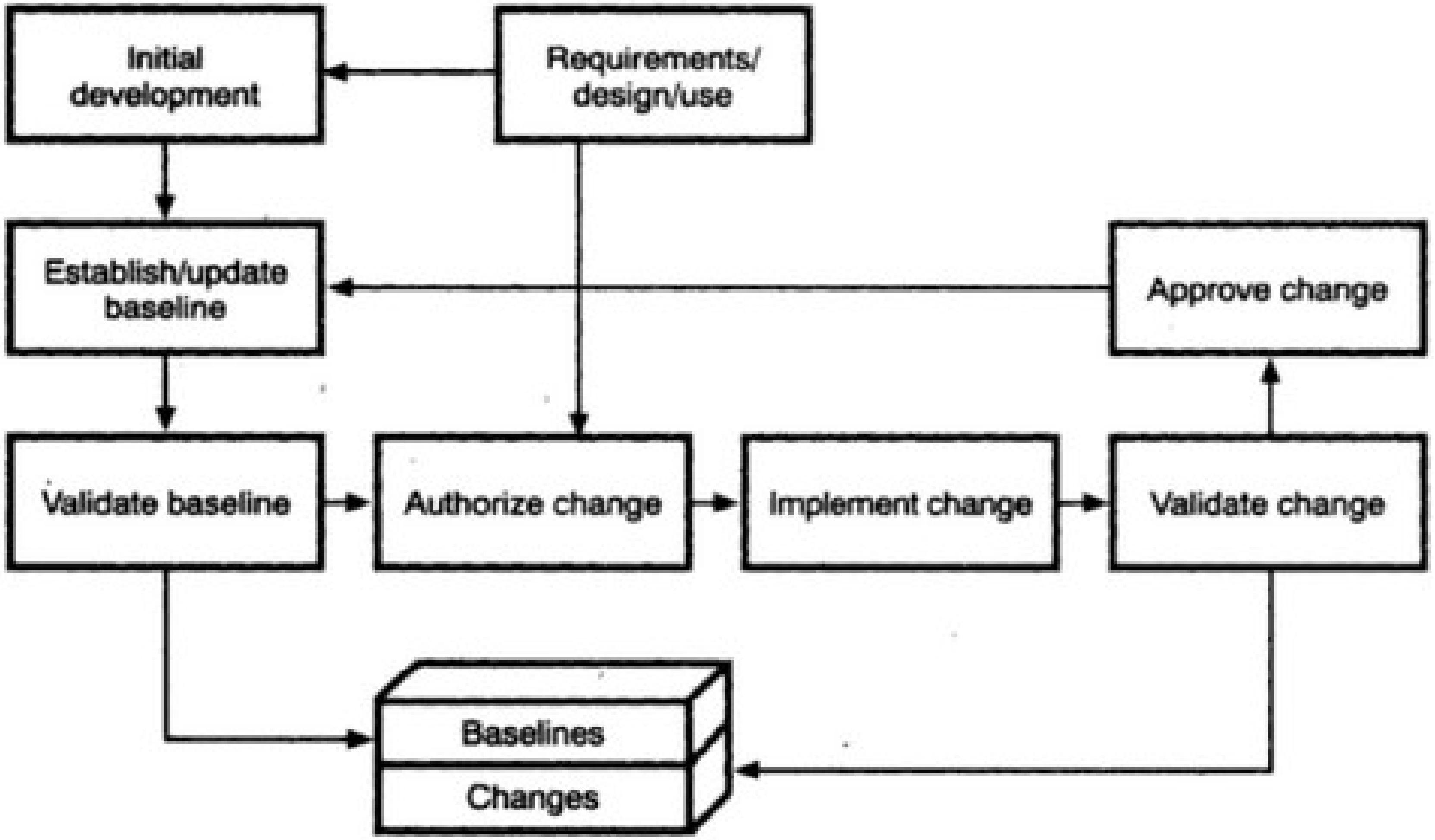
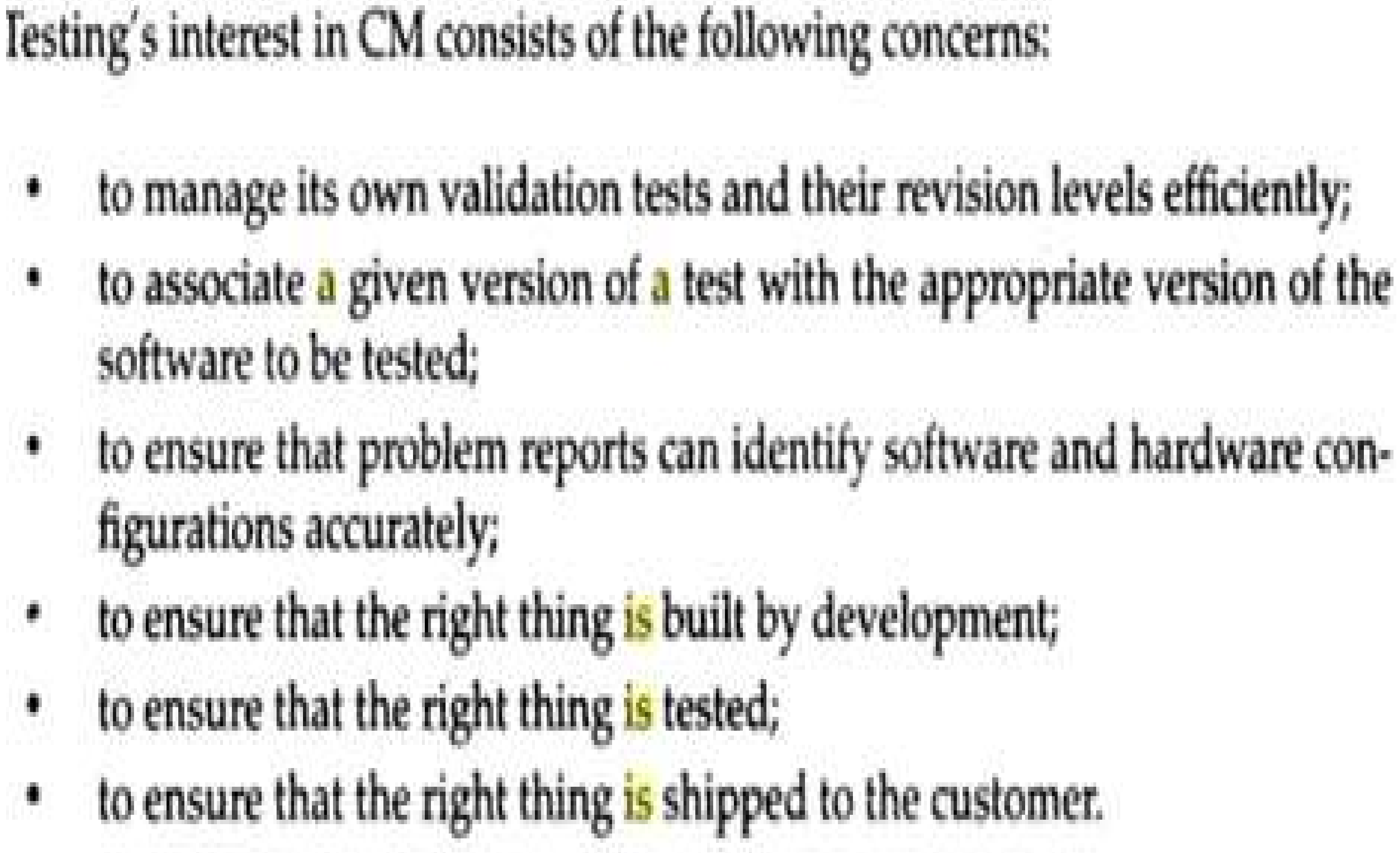


Figure 6.2 An overview of CM. (© 1993, 1994 Software Development Technologies)

Testing Interest in CM



Testing's interest in CM consists of the following concerns:

- to manage its own validation tests and their revision levels efficiently;
- to associate a given version of a test with the appropriate version of the software to be tested;
- to ensure that problem reports can identify software and hardware configurations accurately;
- to ensure that the right thing is built by development;
- to ensure that the right thing is tested;
- to ensure that the right thing is shipped to the customer.

Standards

Why standards? The use of standards simplifies communication, promotes consistency and uniformity, and eliminates the need to invent yet another (often different and even incompatible) solution to the same problem.

IEEE/ANSI standards

Many key standards relating to software testing are generated by the Institut
of Electrical and Electronics Engineers (IEEE).

Key US software testing standards

- IEEE Standard for Software Test Documentation, Reaff. 1991 (IEEE/ ANSI Std 829-1983)
- IEEE Standard for Software Unit Testing, Reaff. 1993 (IEEE/ANSI Std 1008-1987)

Other standards related to software testing

- IEEE Standard for Software Verification and Validation Plans, Reaff. 1992 (IEEE/ANSI Std 1012-1986)
- IEEE Standard for Software Reviews and Audits (IEEE/ANSI Std 1028-1988)

- IEEE Standard for Software Quality Assurance Plans (IEEE/ANSI Std 730-1989)
- IEEE Standard Glossary of Software Engineering Terminology (IEEE/ANSI Std 610.12-1990)

IEEE uses one of three designations in every document title:

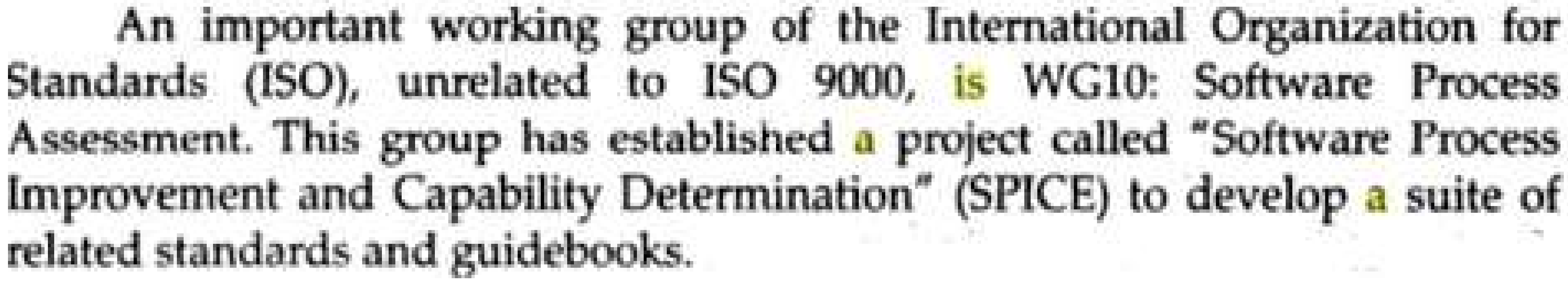
(1) *Standard* means "must use";

(2) *Recommended practice* means "should use";

(3) *Guide* means "use at your discretion".

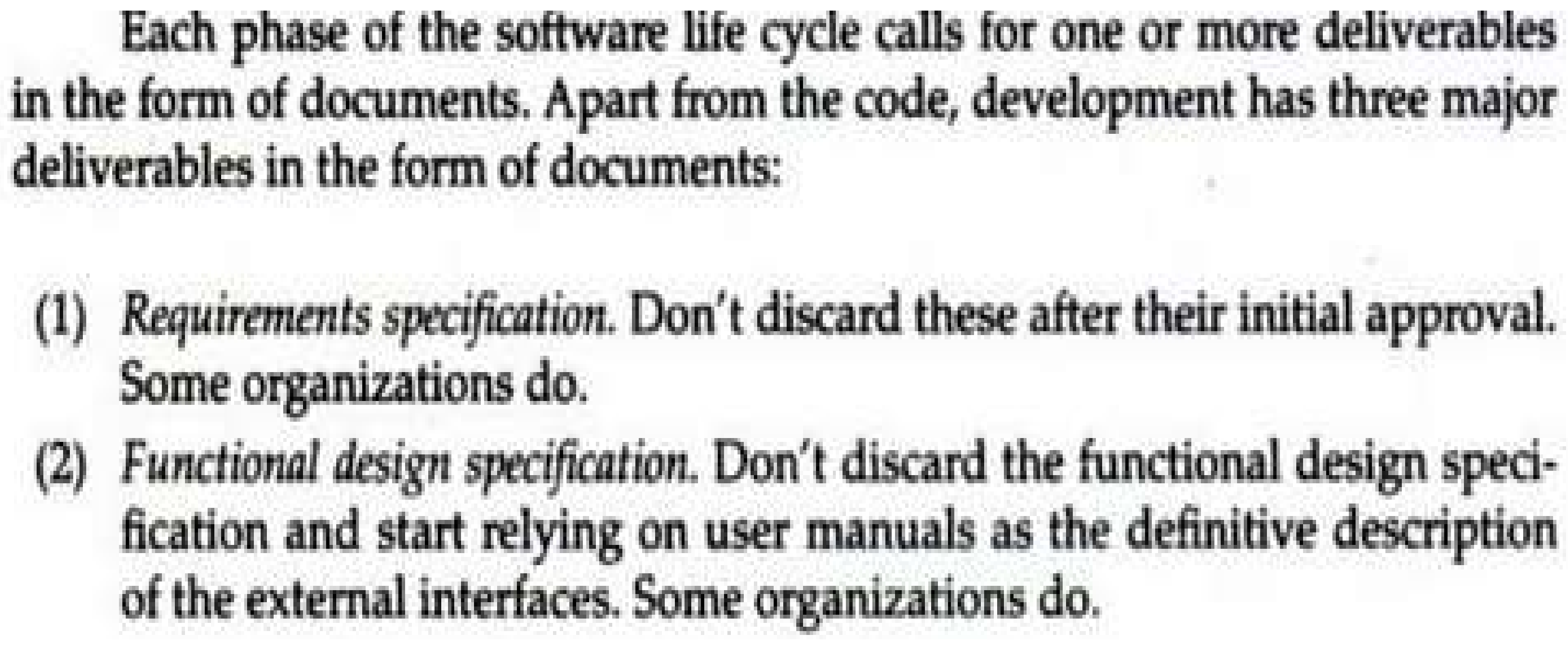
ISO 9000, SPICE and other standards

The ISO 9000 series of standards addresses the quality management system of an organization, and ISO 9001 is the base international standard for quality management. ISO 9000-3 is a guidebook on how ISO 9001 applies to software.


An important working group of the International Organization for Standards (ISO), unrelated to ISO 9000, is WG10: Software Process Assessment. This group has established a project called "Software Process Improvement and Capability Determination" (SPICE) to develop a suite of related standards and guidebooks.

Formal Documents

- Formal document is document of which the form , general content and timing has been agreed in advance. It is deliverable.
- It is document in order to put decision in writing.
- Document is vehicle for stake holder communication.
- Document are delivered by the development and testing group.

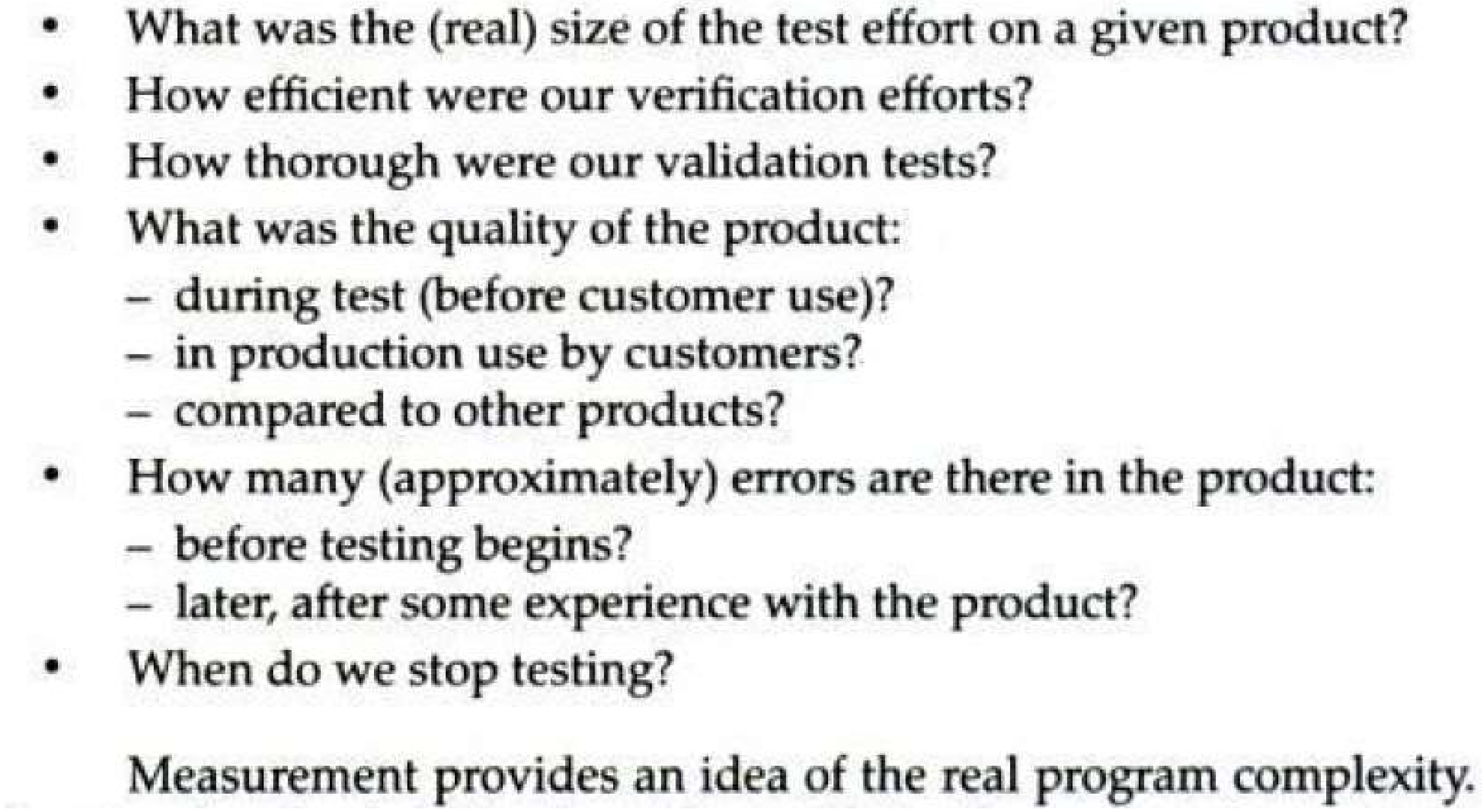


Each phase of the software life cycle calls for one or more deliverables in the form of documents. Apart from the code, development has three major deliverables in the form of documents:

(1) *Requirements specification.* Don't discard these after their initial approval. Some organizations do.

(2) *Functional design specification.* Don't discard the functional design specification and start relying on user manuals as the definitive description of the external interfaces. Some organizations do.

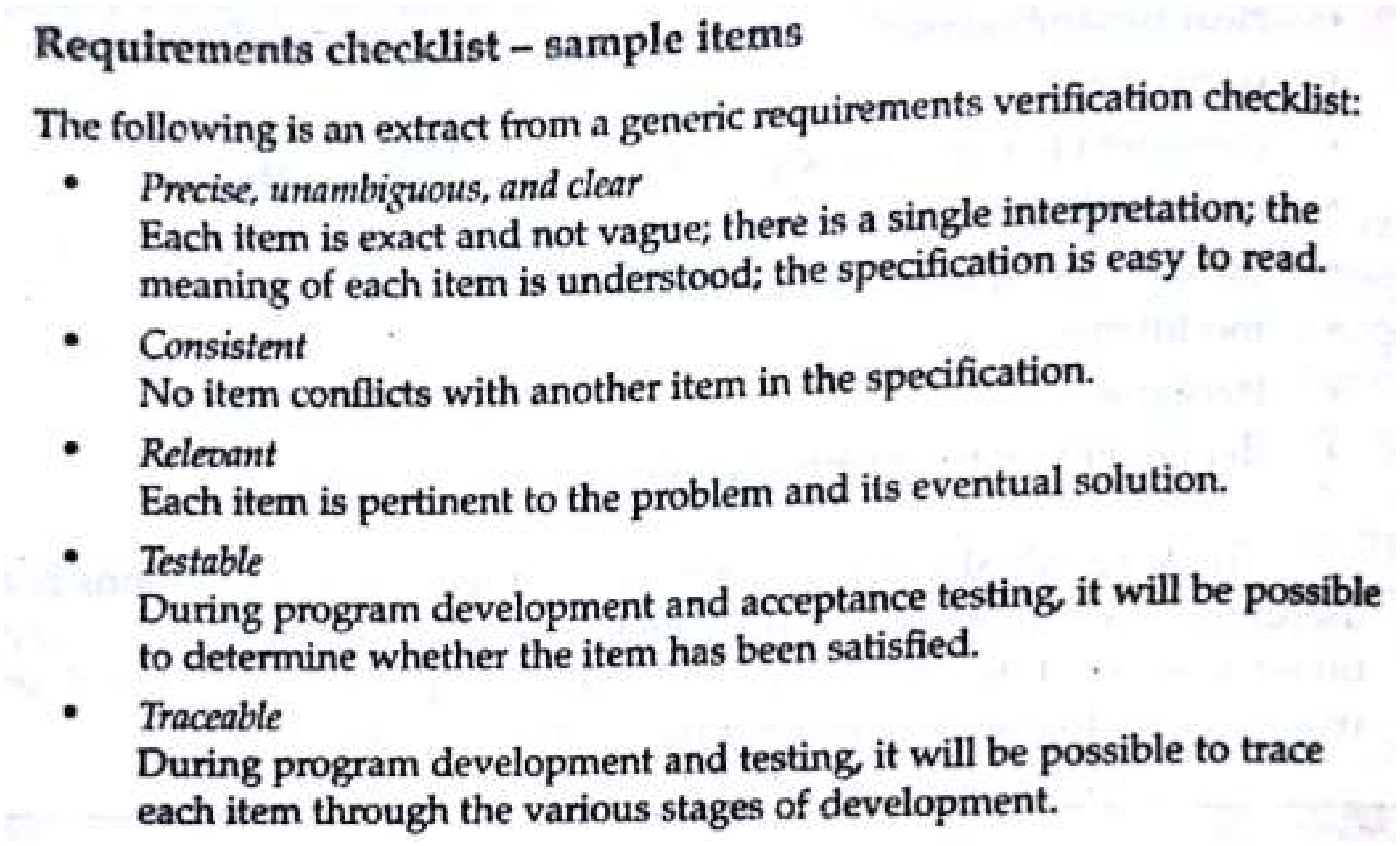
Measurement



- What was the (real) size of the test effort on a given product?
- How efficient were our verification efforts?
- How thorough were our validation tests?
- What was the quality of the product:
  - during test (before customer use)?
  - in production use by customers?
  - compared to other products?
- How many (approximately) errors are there in the product:
  - before testing begins?
  - later, after some experience with the product?
- When do we stop testing?

Measurement provides an idea of the real program complexity.

- Measurement and **tracking incident reports**
- Provides a leading indicators of **product quality**
- Provides significant criteria for **readiness**
- Correlates to user **satisfaction with product**
- Serves predictor for **remaining error**
- Provide measure of **product quality relative to other product**
- Provide measure of **testing efficiency**

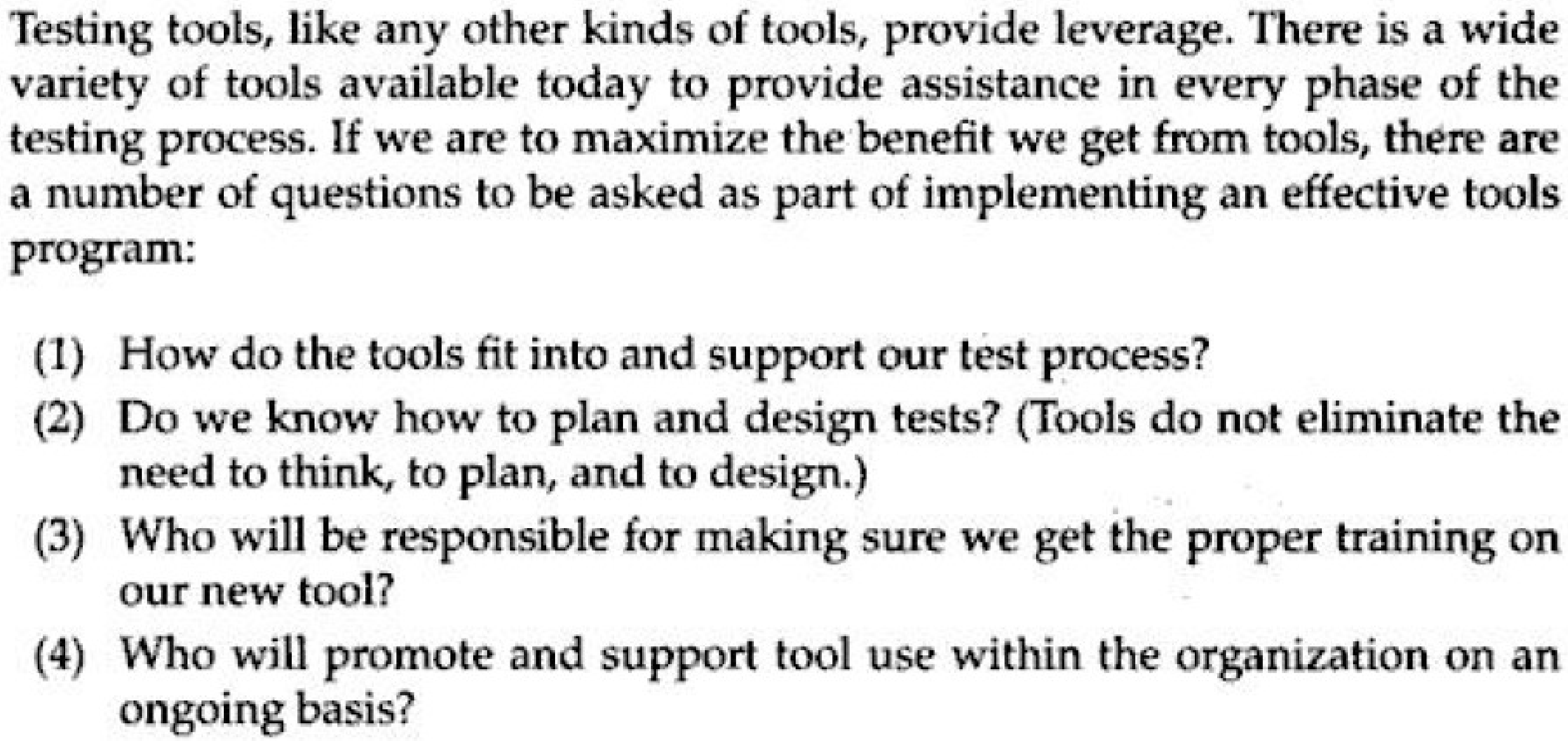


Requirements checklist – sample items

The following is an extract from a generic requirements verification checklist:

- *Precise, unambiguous, and clear*
  Each item is exact and not vague; there is a single interpretation; the meaning of each item is understood; the specification is easy to read.
- *Consistent*
  No item conflicts with another item in the specification.
- *Relevant*
  Each item is pertinent to the problem and its eventual solution.
- *Testable*
  During program development and acceptance testing, it will be possible to determine whether the item has been satisfied.
- *Traceable*
  During program development and testing, it will be possible to trace each item through the various stages of development.

TOOLS

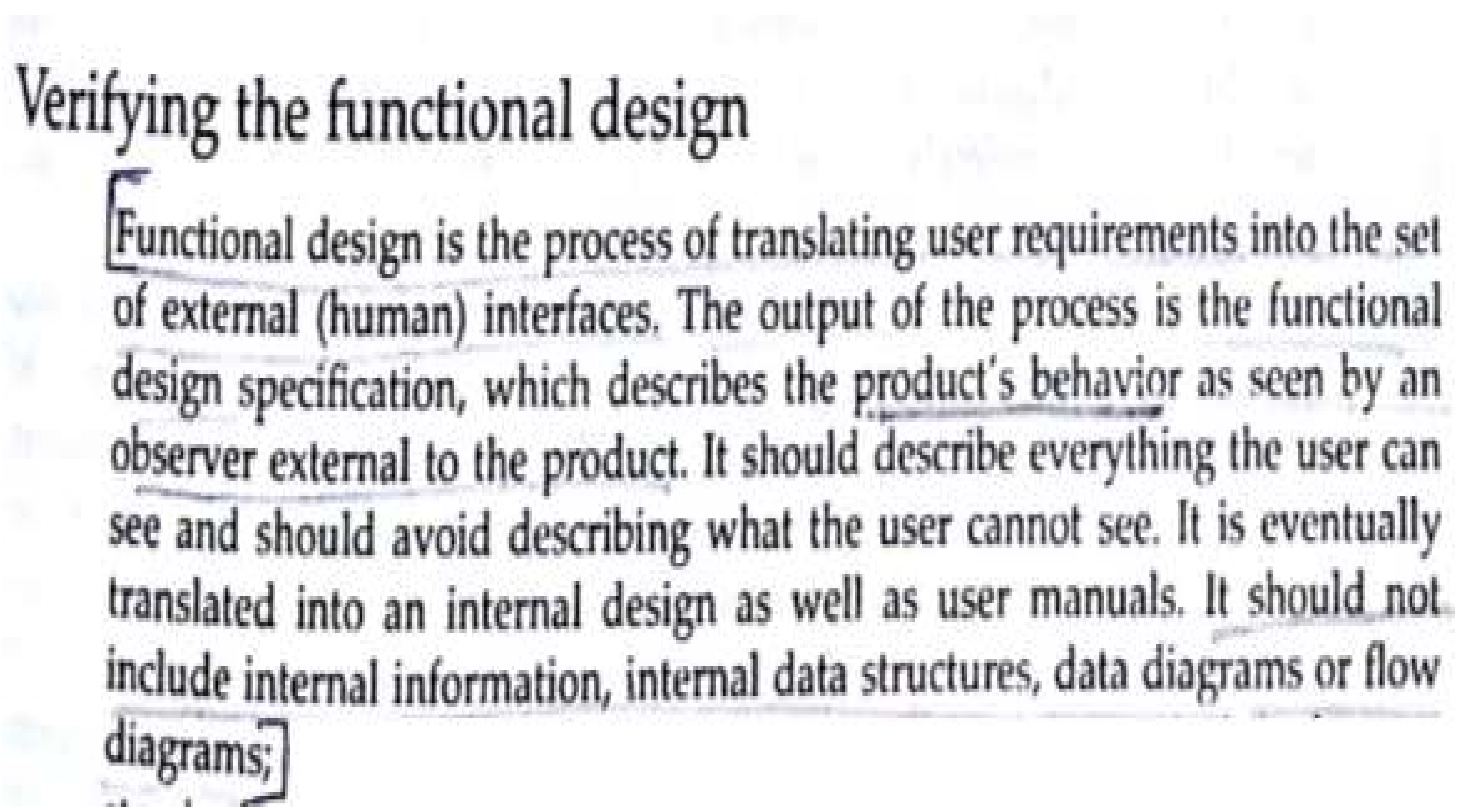


Testing tools, like any other kinds of tools, provide leverage. There is a wide variety of tools available today to provide assistance in every phase of the testing process. If we are to maximize the benefit we get from tools, there are a number of questions to be asked as part of implementing an effective tools program:

(1) How do the tools fit into and support our test process?
(2) Do we know how to plan and design tests? (Tools do not eliminate the need to think, to plan, and to design.)
(3) Who will be responsible for making sure we get the proper training on our new tool?
(4) Who will promote and support tool use within the organization on an ongoing basis?

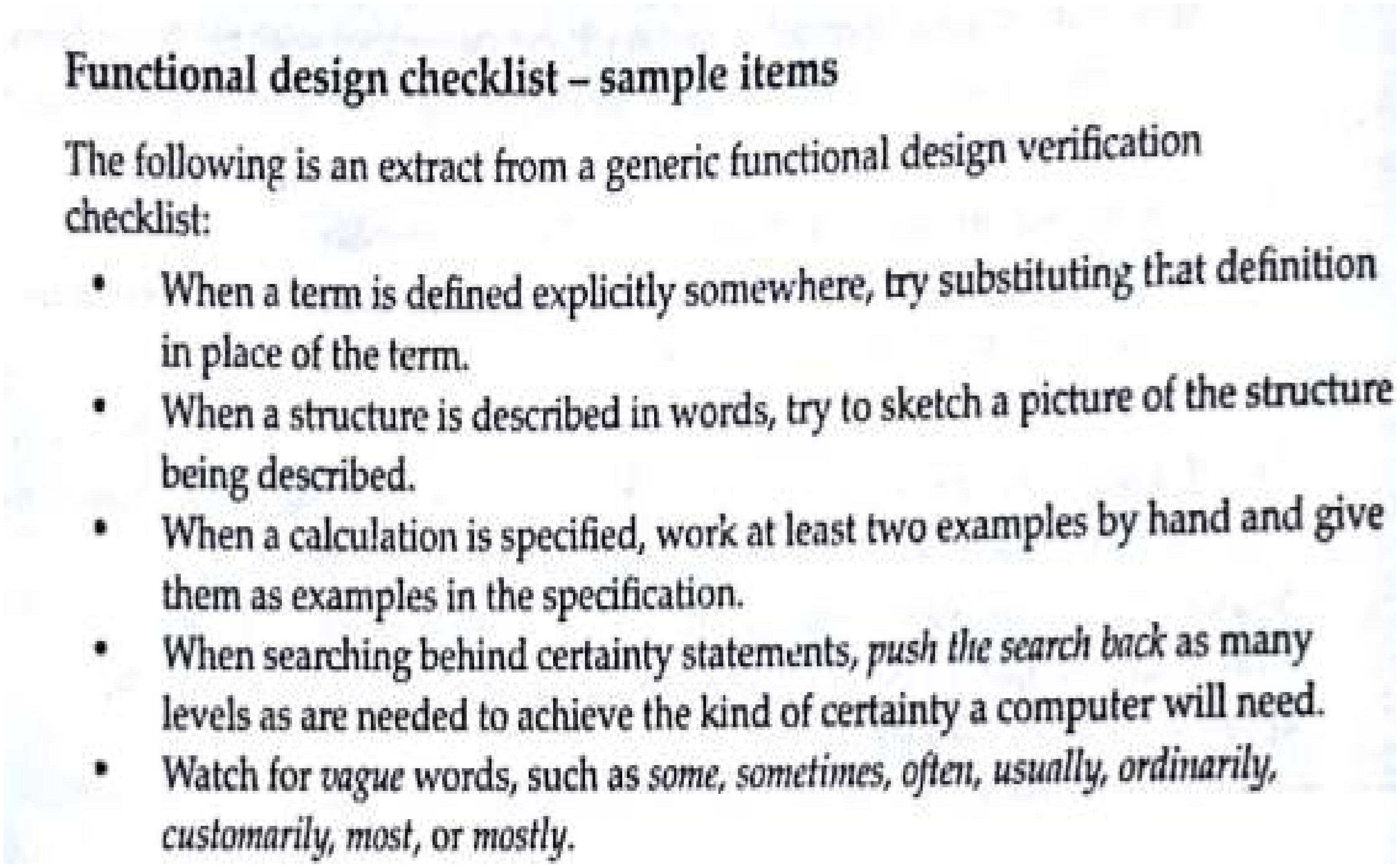
Tools can be categorized in a number of ways:

- by the testing activity or task in which they are employed (e.g., code verification, test planning, test execution);
- by descriptive functional keyword, in other words the specific function performed by the tool (e.g., capture/reply, logic coverage, comparator);
- by major areas of classification, in other words a small number of high-level classes or groupings of tools, each class containing tools that are similar in function or other characteristic (e.g., test management, static analysis, simulator).



Verifying the functional design

Functional design is the process of translating user requirements into the set of external (human) interfaces. The output of the process is the functional design specification, which describes the product's behavior as seen by an observer external to the product. It should describe everything the user can see and should avoid describing what the user cannot see. It is eventually translated into an internal design as well as user manuals. It should not include internal information, internal data structures, data diagrams or flow diagrams;



Functional design checklist – sample items

The following is an extract from a generic functional design verification checklist:

- When a term is defined explicitly somewhere, try substituting that definition in place of the term.
- When a structure is described in words, try to sketch a picture of the structure being described.
- When a calculation is specified, work at least two examples by hand and give them as examples in the specification.
- When searching behind certainty statements, *push the search back* as many levels as are needed to achieve the kind of certainty a computer will need.
- Watch for *vague* words, such as *some, sometimes, often, usually, ordinarily, customarily, most*, or *mostly*.